FUJITSU SEMICONDUCTOR

CM25-00324-1E

CONTROLLER MANUAL

F^2MC^{TM}-8L/8FX FAMILY

$SOFTUNE^{TM}$ WORKBENCH

USER'S MANUAL

FUJITSU

F^2MC^{TM}-8L/8FX FAMILY

SOFTUNE^{TM} WORKBENCH

USER'S MANUAL

FUJITSU LIMITED

PREFACE

■ What is the SOFTUNE Workbench?

SOFTUNE Workbench is support software for developing programs for the FR-V, FR, F^2MC-8L/8FX, and F^2MC-8L families of microprocessors / microcontrollers.

It is a combination of a development manager, simulator debugger, emulator debugger, monitor debugger, and an integrated development environment for efficient development.

■ Purpose of this manual and target readers

This manual explains functions of SOFTUNE Workbench.

This manual is intended for engineers designing several kinds of products using SOFTUNE Workbench.

■ Trademarks

SOFTUNE is a trademark of FUJITSU LIMITED.

FR is the abbreviation of FUJITSU RISC Controller and a product of FUJITSU LIMIITED.

F^2MC is the abbreviation of FUJITSU Flexible Microcontroller and a trademark of FUJITSU LIMITED.

Windows is registered trademarks of Microsoft Corporation in the U.S. and other countries.

■ Organization of Manual

This manual consists of 2 chapters.

CHAPTER 1 Basic Functions

This chapter describes the basic functions on the SOFTUNE Workbench.

CHAPTER 2 Dependence Function

This chapter describes the functions dependent on F^2MC-8L/8FX family MCU.

CHAPTER 1 Basic Functions

This chapter describes the basic functions on the SOFTUNE Workbench.

1.1 Workspace Management Function

This section explains the workspace management function of SOFTUNE Workbench.

■ Workspace

SOFTUNE Workbench uses workspace as a container to manage two or more projects including subprojects.

For example, a project that creates a library and a project that creates a target file using the project can be stored in one workspace.

■ Workspace Management Function

To manage two or more projects, workspace manages the following information:

- Project
- Active project
- Subproject

■ Project

The operation performed in SOFTUNE Workbench is based on the project. The project is a set of files and procedures necessary for creation of a target file. The project file contains all data managed by the project.

■ Active Project

The active project is basic to workspace and undergoes [Make], [Build], [Compile/Assemble], [Start Debug], and [Update Dependence] in the menu. [Make], [Build], [Compile/Assemble], and [Update Dependence] affect the subprojects within the active project.

If workspace contains some project, it always has one active project.

■ Subproject

The subproject is a project on which other projects depend. The target file in the subproject is linked with the parent project of the subproject in creating a target file in the parent project.

This dependence consists of sharing target files output by the subproject, so a subproject is first made and built. If making and building of the subproject is unsuccessful, the parent project of the subproject will not be made and built.

The target file in the subproject is however not linked with the parent project when:

- An absolute format (ABS)-type project is specified as a subproject.
- A library (LIB)-type project is specified as a subproject.

■ Restrictions on Storage of Two or More Projects

Only one REALOS-type project can be stored in one workspace.

1.2 Project Management Function

This section explains the project management function of SOFTUNE Workbench.

■ Project Management Function

The project manages all information necessary for development of a microcontroller system. Especially, its major purpose is to manage information necessary for creation of a target file.

The project manages the following information:

- Project configuration
- Active project configuration
- Information on source files, include files, other object files, library files
- Information on tools executed before and after executing language tools (customize build function)

■ Project format

The project file supports two formats: a 'workspace project format,' and an 'old project format.'

The differences between the two formats are as follows:

Workspace project format

- Supports management of two or more project configurations
- Supports use of all macros usable in manager
- Does not support early Workbench versions *

Old project format

- Supports management of just one project configuration
- Limited number of macros usable in manager
  For details, see Section 1.11 Macro Descriptions Usable in Manager.
- Supports early Workbench versions *

When a new project is made, the workspace project format is used.

When using an existing project, the corresponding project format is used.

If a project made by an early Workbench version * is used, a dialog asking whether to convert the file to the workspace project format is displayed. For details, refer to Section 2.13 of SOFTUNE WORKBENCH Operation Manual.

To open a project file in the workspace project format with an early Workbench version *, it is necessary to convert the file to the old project format. For saving the file in other project formats, refer to Section 4.2.7 Save As of SOFTUNE WORKBENCH Operation Manual.

*: F^{2}MC-8L: V30L26 or earlier

■ Project Configuration

The project configuration is a series of settings for specifying the characteristics of a target file, and making, building, compiling and assembling is performed in project configurations.

Two or more project configurations can be created in a project. The default project configuration name is Debug. A new project configuration is created on the setting of the selected existing project configuration. In the new project configuration, the same files as those in the original project configuration are always used.

By using the project configuration, the settings of programs of different versions, such as the optimization level of a compiler and MCU setting, can be created within one project.

In the project configuration, the following information is managed:

- Name and directory of target file
- Information on options of language tools to create target file by compiling, assembling and linking source files
- Information on whether to build file or not
- Information on setting of debugger to debug target file

■ Active Project Configuration

The active project configuration at default undergoes [Make], [Build], [Compile/Assemble], [Start Debug], and [Update Dependence].

The setting of the active project configuration is used for the file state displayed in the project window and includes files detected in the Dependencies folder.

Note:

If a macro function newly added is used in old project format, the macro description is expanded at the time of saving in old project format. For the macro description newly added, refer to Section 1.11 Macro Descriptions Usable in Manager.

1.3 Project Dependence

This section explains the project dependence of SOFTUNE Workbench.

■ Project Dependence

If target files output by other projects must be linked, a subproject is defined in the project required in [Project Dependence] in the [Project] menu. The subproject is a project on which other projects depend.

By defining project dependence, a subproject can be made and built to link its target file before making and building the parent project.

The use of project dependence enables simultaneous making and building of two or more projects developed in one workspace.

A project configuration in making and building a subproject in [Project Configuration]-[Build Configuration] in the [Project] menu can be specified.

1.4 Make/Build Function

This section explains the make/build function of SOFTUNE Workbench.

■ Make Function

Make function generates a target file by compiling/assembling only updated source files from all source files registered in a project, and then joining all required object files.

This function allows compiling/assembling only the minimum of required files. The time required for generating a target file can be sharply reduced, especially, when debugging.

For this function to work fully, the dependence between source files and include files should be accurately grasped. To do this, SOFTUNE Workbench has a function for analyzing include dependence. For details, see Section 1.5 Include Dependencies Analysis Function.

■ Build Function

Build function generates a target file by compiling/assembling all source files registered with a project, regardless of whether they have been updated or not, and then by joining all required object files. Using this function causes all files to be compiled/assembled, resulting in the time required for generating the target file longer. Although the correct target file can be generated from the current source files.

The execution of Build function is recommended after completing debugging at the final stage of program development.

Note:

When executing the Make function using a source file restored from backup, the integrity between an object file and a source file may be lost. If this happens, executing the Build function again.

1.4.1 Customize Build Function

This section describes the SOFTUNE Workbench to set the Customize Build function.

■ Customize Build function

In SOFTUNE Workbench, different tools can be operated automatically before and after executing the Assembler, Compiler, Linker, Librarian, Converter, or Configurator started at Compile, Assemble, Make, or Build.

The following operations can be performed automatically during Make or Build using this function:

- Starting the syntax check before executing the Compiler,
- After executing the Converter, starting the S-format binary Converter (m2bs.exe) and converting Motorola S-format files to binary format files.

■ Setting Options

An option follows the tool name to start a tool from SOFTUNE Workbench. The options include any file name and tool-specific options. SOFTUNE Workbench has the macros indicating that any file name and tool-specific options are specified as options.

If any character string other than parameters is specified, it is passed directly to the tool as it is. For details about the parameters, see Section 1.11 Macro Descriptions Usable in Manager.

■ Macro List

The Setup Customize Build dialog provides a macro list for macro input. The build file, load module file, project file submenus indicate their sub-parameters specified.

The environment variable brackets must have any item; otherwise, resulting in an error.

Table 1.4-1 Macro List

Macro List	Macro Name
Build file	%(FILE)
Load module file	%(LOADMODULEFILE)
Project file	%(PRJFILE)
Workspace file	%(WSPFILE)
Project directory	%(PRJPATH)
Target file directory	%(ABSPATH)
Object file directory	%(OBJPATH)
List file directory	%(LSTPATH)
Project construction name	%(PRJCONFIG)
Environment variable	%(ENV[])
Temporary file	%(TEMPFILE)

Note:

When checking [Use the Output window], note the following:

- Once a tool is activated, Make/Build is suspended until the tool is terminated.
- The Output window must not be used with a tool using a wait state for user input while the tool is executing. The user can not perform input while the Output window is in use, so the tool cannot be terminated. To forcibly terminate the tool, select the tool on the Task bar and input Control - C, or Control - Z.

1.5 Include Dependencies Analysis Function

This section describes the function of the Include Dependencies Analysis of SOFTUNE Workbench.

■ Analyzing Include Dependencies

A source file usually includes some include files. When only an include file has been modified leaving a source file unchanged, SOFTUNE Workbench cannot execute the Make function unless it has accurate and updated information about which source file includes which include files.

For this reason, SOFTUNE Workbench has built-in Include Dependencies Analysis function. This function can be activated by selecting the [Project] -[Include Dependencies] command. By using this function, users can know the exact dependencies, even if an include file includes another include file.

SOFTUNE Workbench automatically updates the dependencies of the compiled/assembled files.

Note:

When executing the [Project] - [Include Dependencies] command, the Output window is redrawn and replaced by the dependencies analysis result.

If the contents of the current screen are important (error message, etc.), save the contents to a file and then execute the Include Dependencies command.

1.6 Functions of Setting Tool Options

This section describes the functions to set options for the language tools activated from SOFTUNE Workbench.

■ Function of Setting Tool Options

To create a desired target file, it is necessary to specify options for the language tools such as a compiler, assembler, and linker. SOFTUNE Workbench stores and manages the options specified for each tool in project configurations.

Tool options include the options effective for all source files (common options) and the options effective for specific source files (individual options). For details about the option setting, refer to Section 4.5.5 Setup Project of SOFTUNE WORKBENCH Operation Manual.

- Common options

  These options are effective for all source files (excluding those for which individual options are specified) stored in the project.

- Individual options

  These options are compile/assemble options effective for specific source files. The common options specified for source files for which individual options are specified become invalid.

■ Tool Options

In SOFTUNE Workbench, the macros indicating that any file name and directory name are specified as options.

If any character string other than parameters is specified, it is passed directly to the tool. For details about the parameters, see Section 1.11 Macro Descriptions Usable in Manager. For details about the tool options for each tool, see the manual of each tool.

1.7 Error Jump Function

This section describes the error jump function in SOFTUNE Workbench.

■ Error Jump Function

When an error, such as a compile error occurs, double-clicking the error message displayed in the Output window, opens the source file where the error occurred, and automatically moves the cursor to the error line. This function permits efficient removal of compile errors, etc.

The SOFTUNE Workbench Error Jump function analyzes the source file names and line number information embedded in the error message displayed in the Output window, opens the matching file, and jumps automatically to the line.

The location where a source file name and line number information are embedded in an error message, varies with the tool outputting the error.

An error message format can be added to an existing one or modified into an new one. However, the modify error message formats for pre-installed Fujitsu language tools are defined as part of the system, these can not be modified.

A new error message format should be added when working the Error Jump function with user registered tool. To set Error Jump, execute the [Setup] - [Error Jump Setting] command.

■ Syntax

An error message format can be described in Syntax. SOFTUNE Workbench uses macro descriptions as shown in the Table 1.7-1 to define such formats.

To analyze up to where %f, %h, and %* continue, SOFTUNE Workbench uses the character immediately after the above characters as a delimiter. Therefore, in Syntax, the description until a character that is used as a delimiter re-appears, is interpreted as a file name or a keyword for help, or is skipped over. To use % as a delimiter, describe as %%. The [%char] macro skips over as long as the specified character continues in parentheses. To specify "]" as a skipped character, describe it as "\]". Blank characters in succession can be specified with a single blank character.

Table 1.7-1 List of Special Characters String for Analyzing Error Message

Parameter	Semantics
%f	Interpret as source file name and inform editor.
%l	Interpret as line number and inform editor.
%h	Become keyword when searching help file.
%*	Skip any desired character.
%[char]	Skip as long as characters in [ ] continues.

[Example]

***   %f(%l)   %h: or, %[*]   %f(%l)   %h:

The first four characters are "***   ", followed by the file name and parenthesized line number, and then the keyword for help continues after one blank character.

This represents the following message:

***C:\Sample\sample.c(100)   E4062C: Syntax Error: near /int.

■ Reference Section

Setup Error Jump

1.8 Editor Functions

This section describes the functions of the SOFTUNE Workbench built-in standard editor.

■ Standard Editor

SOFTUNE Workbench has built-in editor called the standard editor. The standard editor is activated as the Edit window in SOFTUNE Workbench. As many Edit windows as are required can be opened at one time.

The standard editor has the following functions in addition to regular editing functions.

- **Keyword marking function in C/C++/assembler source file**

  Displays reserved words, such as if and for, in different color

- **Error line marking function**

  The error line can be viewed in a different color, when executing Error Jump.

- **Tag setup function**

  A tag can be set on any line, and instantaneously jumps to the line. Once a tag is set, the line is displayed in a different color.

- **Ruler, line number display function**

  The Ruler is a measure to find the position on a line; it is displayed at the top of the Edit window. A line number is displayed at the left side of the Edit window.

- **Automatic indent function**

  When a line is inserted using the Enter key, the same indent (indentation) as the preceding line is set automatically at the inserted line. If the space or tab key is used on the preceding line, the same use is set at the inserted line as well.

- **Function to display, Blank, Line Feed code, and Tab code**

  When a file includes a Blank, Line Feed code, and Tab code, these codes are displayed with special symbols.

- **Undo function**

  This function cancels the preceding editing action to restore the previous state. When more than one character or line is edited, the whole portion is restored.

- **Tab size setup function**

  Tab stops can be specified by defining how many digits to skip when Tab codes are inserted. The default is 8.

- **Font changing function**

  The font size for character string displayed in the Edit window can be selected.

■ Reference section

Edit Window (The Standard Editor)

1.9 Storing External Editors

This section describes the function to set an external editor to SOFTUNE Workbench.

■ External Editor

SOFTUNE Workbench has built-in standard editor, and use of this standard editor is recommended. However, another accustomed editor can be used, with setting it, instead of an edit window. There is no particular limit on which editor can be set, but some precautions (below) may be necessary. Use the [Setup] - [Editor setting] command to set an external editor.

■ Precautions

- **Error jump function**

  The Error Jump cannot move the cursor to an error line if the external editor does not have a function to specify the cursor location when activated the external editor.

- **File save at compiling/assembling**

  SOFTUNE Workbench cannot control an external editor. Always save the file you are editing before compiling/assembling.

■ Setting Options

When activating an external editor from SOFTUNE Workbench, options must be added immediately after the editor name. The names of file to be opened by the editor and the initial location of the cursor (the line number) can be specified. SOFTUNE Workbench has a set of special parameters for specifying any file name and line number, as shown in the Table 1.9-1 . If any other character string are described by these parameters, such characters string are passed as it is to the editor.

%f (File name) is determined as follows:

1. If the focus is on the project window, and if a valid file name is selected, the selected file name becomes the file name.
2. When a valid file name cannot be acquired by the above procedure, the file name with a focus in built-in editor becomes the file name.

%x (project path) is determined as follows:

1. If a focus is on the project window and a valid file name is selected, the project path is a path to the project in which the file is stored.
2. If no path is obtained, the project path is a path to the active project.

Also file name cannot be given double-quotes in the expansion of %f macros.

Therefore, it is necessary for you to provide double-quotes for %f. Depending on the editor, there are line numbers to which there will be no correct jump if the entire option is not given double-quotes.

Table 1.9-1 List of Special Characters for Analyzing Error Message

Parameter	Semantics
%%	Means specifying % itself
%f	Means specifying file name
%l	Means specifying line number
%x	Means specifying project path

■ Example of Optional Settings

Table 1.9-2 Parameters Used in Option Setups (For External Editors)

Editor name	Argument
WZ Editor V4.0	%f /j%l
MIFES V1.0	%f+%l
UltraEdit32	%f/%l/1
TextPad32	%f(%l)
PowerEDITOR	%f -g%l
Codewright32	%f -g%l
Hidemaru for Win3.1/95	/j%l:1 %f
ViVi	/line=%l %f

■ Reference Section

Editor Setup

Note:

Regarding execution of error jump in Hidemaru:

To execute error jump in Hidemaru used as an external editor, use the [Others] - [Operating Environment] - [Exclusive Control] command, and then set "When opening the same file in Hidemaru" and "Opening two identical files is inhibited".

1.10 Storing External Tools

This section describes the function to set an external tool to SOFTUNE Workbench.

■ External Tools

A non-standard tool not attached to SOFTUNE Workbench can be used by setting it as an external tool and by calling it from SOFTUNE Workbench. Use this function to coordinate with a source file version management tool.

If a tool set as an external tool is designed to output the execution result to the standard output and the standard error output through the console application, the result can be specified to output the SOFTUNE Workbench Output window. In addition, the allow description of additional parameters each time the tool is activated.

To set an external tool, use the [Setup] - [Setting Tool] command.

To select the title of a set tool, use the [Setup] - [Activating Tool] command.

■ Setting Options

When activating an external tool from SOFTUNE Workbench, options must be added immediately after the external tool name. Specify the file names and unique options, etc.

SOFTUNE Workbench has a set of special parameters for specifying any file name and unique tool options.

If any characters string described other than these parameters, such characters string are passed as it is to the external tool.

For details about the parameters, see Section 1.11 Macro Descriptions Usable in Manager.

Note:

When checking [Use the Output window], note the following:

1. Once a tool is activated, neither other tools nor the compiler/assembler can be activated until the tool is terminated.
2. The Output window must not be used with a tool using a wait state for user input while the tool is executing. The user cannot perform input while the Output window is in use, so the tool cannot be terminated. To forcibly terminate the tool, select the tool on the Task bar and input Control - C, or Control - Z.

■ Reference Section

Setting Tools

Starting Tools

1.11 Macro Descriptions Usable in Manager

This section explains the macro descriptions that can be used in the manager of SOFTUNE Workbench.

■ Macros

SOFTUNE Workbench has special parameters indicating that any file name and tool-specific options are specified as options.

The use of these parameters as tool options eliminates the need for options specified each time each tool is started.

The type of macro that can be specified and macro expansion slightly vary depending on where to describe macros. The macros usable for each function are detailed below. For the macros that can be specified for "Error Jump" and "External Editors" see Sections "1.7 Error Jump Function" and "1.9 Storing External Editors".

■ Macro List

The following is a list of macros that can be specified in SOFTUNE Workbench.

The macros usable for each function are listed below.

- External tools:	Table 1.11-1 and Table 1.11-2
- Customize build:	Table 1.11-1 and Table 1.11-2
- Tool options:	Table 1.11-2

The directory symbol \ is added to the option directories in Table 1.11-1 but not to the macro directories in Table 1.11-2 .

The sub-parameters in Table 1.11-3 can be specified in %(FILE), %(LOADMODULEFILE), %(PRJFILE), and %(WSPFILE).

The sub-parameter is specified in the form of %(PRJFILE[PATH]).

If the current directory is on the same drive, the relative path is used. The current directory is the workspace directory for %(PRJFILE) and %(WSPFILE), and the project directory for other than them.

Table 1.11-1 List of Macros That Can Be Specified 1

Parameter	Meaning
%f	Passed as full-path name of file. [*1]
%F	Passed as main file name of file. [*1]
%d	Passed as directory of file. [*1]
%e	Passed as extension of file. [*1]
%a	Passed as full-path name of load module file.
%A	Passed as main file name of load module file. [*2]
%D	Passed as directory of load module file. [*2]
%E	Passed as extension of load module file. [*2]
%x	Passed as directory of project file. [*2]
%X	Passed as main file name of project file. [*2]
%%	Passed as %.

Table 1.11-2 List of Macros That Can Be Specified 2

Parameter	Meaning
%(FILE)	Passed as full-path name of file. [*1]
%(LOADMODULEFILE)	Passed as full-path name of load module file. [*2]
%(PRJFILE)	Passed as full-path name of project file. [*2]
%(WSPFILE)	Passed as full-path name of workspace file. [*3]
%(PRJPATH)	Passed as directory of project file. [*2]
%(ABSPATH)	Passed as directory of target file. [*2]
%(OBJPATH)	Passed as directory of object file. [*2]
%(LSTPATH)	Passed as directory of list file. [*2]
%(PRJCONFIG)	Passed as project configuration name. [*2, *3]
%(ENV [Environment variable])	Environment variable value specified in environment variable brackets is passed.
%(TEMPFILE)	Temporary file is created and its full-path name is passed. [*4]

*1:The macros are determined as follows:

- Customize build

1. Source file before and after executing compiler and assembler
2. Target file before and after executing linker, librarian and converter
3. Configuration file before and after executing configuration

- Tool options

  Null character

- Others

1. File as focus is on project window and valid file name is selected
2. File on which focus is in internal editor as no valid file name can be obtained in 1
3. Null character if no valid file name can be obtained

*2:The macros are determined as follows:

- Customize build and tool options

  Information on configuration of project under building, making, compiling and assembling

- Others

1. Information on active configuration of project in which file is stored as focus is on project window and valid file name is selected
2. Information on active configuration of active project if no valid file name can be obtained in 1

*3:Only project files in the workspace project format can be used for macros indicated.
*4:Data in the temporary file can be specified only for customize build.

Table 1.11-3 List of Sub parameters 1

Sub parameter	Meaning
[PATH]	Directory of file
[RELPATH]	Relative path of file
[NAME]	Main file name of file
[EXT]	Extension of file
[SHORTFULLNAME]	Full path name of short file
[SHORTPATH]	Directory of short file
[SHORTNAME]	Main file name of short file
[FOLDER]	Name of folder in which files are stored in project window (Can be specified only in %(FILE).) *

The macro in "*" can be used only the project of workspace project format.

■ Examples of Macro Expansion

If the following workspace is opened, macro expansion is performed as follows:

Workspace	:	C:\Wsp\Wsp.wsp
Active project	:	C:\Wsp\Sample\Sample.prj

Active project configuration - Debug

Object directory	:	C:\Wsp\Sample\Debug\Obj\

Subproject	:	C:\Subprj\Subprj.prj
Active project configuration - Release		
Object directory	:	C:\Subprj\Release\Obj\
Target file	:	C:\Subprj\Release\Abs\Subprj.abs

[Example] Macro expansion in external tools

Focus is on Subprj project file in project window.

%a	:	C:\Subprj\Release\Abs\Subprj.abs
%A	:	SUBPRJ.abs
%D	:	C:\Subprj\Release\Abs\
%E	:	.abs
%(FILE[FOLDER])	:	Source Files\Common
%(PRJFILE)	:	C:Subprj\Subprj.prj

Focus is not in project window.

%a	:	C:\Wsp\Sample\Debug\Abs\Sample.abs
%A	:	Sample.abs
%D	:	C:\Wsp\Sample\Debug\Abs\
%(PRJFILE)	:	C:\Wsp\Sample\Sample.prj

[Example] Macro expansion in customize build

Release configuration of Subprj project is built.

%(FILE)	:	C:\Subprj\LongNameFile.c
%(FILE[PATH])	:	C:\Subprj
%(FILE[RELPATH])	:	.
%(FILE[NAME])	:	LongNameFile
%(FILE[EXT])	:	.c
%(FILE[SHORTFULLNAME])	:	C:\Subprj\LongFi~1.c
%(FILE[SHORTPATH])	:	C:\Subprj
%(FILE[SHORTNAME])	:	LongFi~1
%(PRJFILE[RELPATH])	:	..\Subprj
%(PRJPATH)	:	C:\Subprj
%(OBJPATH)	:	C:\Subprj\Release\Obj
%(PRJCONFIG)	:	Release
%(ENV[FETOOL])	:	C:\SOFTUNE
%(TEMPFILE)	:	C:\Subprj\Release\Opt_fs1056.TMP

[Example] Macro expansion in tool options

Release configuration of Subprj project is built.

%(FILE) :

%(PRJFILE[RELPATH])	:	..\Subprj
%(PRJPATH)	:	C:\Subprj
%(OBJPATH)	:	C:\Subprj\Release\Obj
%(PRJCONFIG)	:	Release
%(ENV[FETOOL])	:	C:\SOFTUNE

1.12 Setting Operating Environment

This section describes the functions for setting the SOFTUNE Workbench operating environment.

■ Operating Environment

Set the environment variables for SOFTUNE Workbench and some basic setting for the workspace.

To set the operating environment, use the **[Setup]-[Development Environment Setting]** command.

- **Environment Variables**

  Environment variables are variables that are referred to mainly using the language tools activated from SOFTUNE Workbench. The semantics of an environment variable are displayed in the lower part of the Setup dialog. However, the semantics are not displayed for environment variables used by tools added later to SOFTUNE Workbench.

  When SOFTUNE Workbench and the language tools are installed in a same directory, it is not especially necessary to change the environment variable setups.

- **Basic setups for workspace**

  The following setups are possible.

  - **Open the previously workspace at start up**

    When starting SOFTUNE Workbench, it automatically opens the last opened workspace.

  - **Display options while compiling/assembling**

    Compile options or assemble options can be viewed in the Output window.

  - **Save dialog before closing workspace**

    Before closing the workspace, a dialog asking for confirmation of whether or not to save the workspace to the file is displayed. If this setting is not made, SOFTUNE Workbench automatically saves the workspace without any confirmation message.

  - **Save dialog at compiling/assembling**

    Before compiling/assembling, a dialog asking for confirmation of whether or not to save a source file that has not been saved is displayed. If this setting is not made, the file is saved automatically at compile/assemble/make/build.

  - **Termination message is highlighted at Make/Build**

    At Compile, Assemble, Make, or Build, the display color of termination messages (Abort, No Error, Warning, Error, Fatal error, or Failing During start) can be changed freely by the user.

■ Reference Section

Development Environment

Note:

Because the environment variables set here are language tools for the SOFTUNE Workbench, the environment variables set on previous versions of SOFTUNE cannot be used. In particular, add the set values of [User Include Directory] and [Library Search Directory] to [Project Settings].

1.13 Debugger Types

This section describes the functions of SOFTUNE Workbench debuggers.

■ Debug Function

SOFTUNE Workbench integrates two types of debugger: a simulator debugger, and emulator debugger. Any one can be selected depending on the requirement.

■ Simulator Debugger

The simulator debugger simulates the MCU operations (executing instructions, memory space, I/O ports, interrupts, reset, etc.) with software to evaluate a program.

It is used for evaluating an uncompleted system and operation of individual units, etc.

■ Emulator Debugger

The emulator debugger is software to evaluate a program by controlling an In-Circuit Emulator (ICE) from a host computer through a communications line (RS-232C, LAN, USB).

Before using this debugger, the ICE must be initialized.

1.14 Memory Operation Functions

This section describes the memory operation functions.

■ Functions for Memory Operations

- **Display/Modify memory data**

  Memory data can be display in the Memory window and modified.

- **Fill**

  The specified memory area can be filled with the specified data.

- **Copy**

  The data in the specified memory area can be copied to another area.

- **Compare**

  The data in the specified source area can be compared with data in the destination area.

- **Search**

  Data in the specified memory area can be searched.

For further details of the above functions, refer to "3.11 Memory Window" in "SOFTUNE WORKBENCH Operation Manual".

- **Display/Modify C/C++ variables**

  The names of variables in a C/C++ source file can be displayed in the Watch window and modified.

- **Setting Watch point**

  By setting a watch point at a specific address, its data can be displayed in the Watch window.

For further details of the above functions, refer to "3.13 Watch Window" in "SOFTUNE WORKBENCH Operation Manual".

1.15 Register Operations

This section describes the register operations.

■ Register Operations

The Register window is opened when the [View] - [Register] command is executed. The register and flag values can be displayed in the Register window.

For further details about modifying the register value and the flag value, refer to "4.4.4 Register" in "SOFTUNE WORKBENCH Operation Manual".

The name of the register and flag displayed in the Register window varies depending on each MCU in use. For the list of register names and flag names for the MCU in use, refer to Appendix A in SOFTUNE WORKBENCH Operational Manual.

■ Reference Section

Register Window

1.16 Line Assembly and Disassembly

This section describes line assembly and disassembly.

■ Line Assembly

To perform line-by-line assembly (line assembly), right-click anywhere in the Disassembly window to display the short-cut menu, and select [Inline Assembly]. For further details about assembly operation, refer to "4.4.3 Assembly" in "SOFTUNE WORKBENCH Operation Manual".

■ Disassembly

To display disassembly, use the [View]-[Assembly] command. By default, disassembly can be viewed starting from the address pointed by the current program counter (PC). However, the address can be changed to any desired address at start-up.

Disassembly for an address outside the memory map range cannot be displayed. If this is attempted, "???" is displayed as the mnemonic.

■ Reference Section

Disassembly Window

1.17 Symbolic Debugging

The symbols defined in a source program can be used for command parameters (address). There are three types of symbols as follows:

- **Global Symbol**
- **Static Symbol within Module (Local Symbol within Module)**
- **Local Symbol within Function**

■ Types of Symbols

A symbol means the symbol defined while a program is created, and it usually has a type. Symbols become usable by loading the debug information file.

Furthermore, for symbol of C/C++, it recognizes the type and executes the command.

There are three types of symbols as follows:

- **Global symbol**

  A global symbol can be referred to from anywhere within a program. In C/C++, variables and functions defined outside a function without a static declaration are in this category. In assembler, symbols with a PUBLIC declaration are in this category.

- **Static symbol within module (Local symbol within module)**

  A static symbol within module can be referred to only within the module where the symbol is defined.

  In C/C++, variables and functions defined outside a function with a static declaration are in this category. In assembler, symbols without a PUBLIC declaration are in this category.

- **Local symbol within function**

  A local symbol within a function exists only in C/C++. A static symbol within a function and an automatic variable are in this category.

  - **Static symbol within function**

    Out of the variables defined in function, those with static declaration.

  - **Automatic variable**

    Out of the variables defined in function, those without static declaration and parameters for the function.

■ Setting Symbol Information

Symbol information in the file is set with the symbol information table by loading a debug information file. This symbol information is created for each module.

The module is constructed for each source file to be compiled in C/C++, in assembler for each source file to be assembled.

The debugger automatically selects the symbol information for the module to which the PC belongs to at abortion of execution (Called "the current module"). A program in C/C++ also has information about which function the PC belongs to.

■ Line Number Information

Line number information is set with the line number information table in SOFTUNE Workbench when a debug information file is loaded. Once registered, such information can be used at anytime thereafter. Line number is defined as follows:

```
[Source File Name]    $Line Number
```

1.17.1 Referring to Local Symbols

This section describes referring to local symbols and Scope.

■ Scope

When a local symbol is referred to, Scope is used to indicate the module and function to which the local symbol to be referred belongs.

SOFTUNE Workbench automatically scopes the current module and function to refer to local symbols in the current module with preference. This is called the Auto-scope function, and the module and function currently being scoped are called the Current Scope.

When specifying a local variable outside the Current Scope, the variable name should be preceded by the module and function to which the variable belongs. This method of specifying a variable is called a symbol path name or a Search Scope.

■ Moving Scope

As explained earlier, there are two ways to specify the reference to a variable: by adding a Search Scope when specifying the variable name, and by moving the Current Scope to the function with the symbol to be referred to. The Current Scope can be changed by displaying the Call Stack dialog and selecting the parent function. For further details of this operation, refer to "4.6.7 Stack" in "SOFTUNE WORKBENCH Operation Manual". Changing the Current Scope as described above does not affect the value of the PC.

By moving the current scope in this way, you can search a local symbol in parent function with precedence.

■ Specifying Symbol and Search Procedure

A symbol is specified as follows:

```
[[Module Name::] [Function Name] \] Symbol Name
```

C++ symbol can be specified as follows with the scope operator:

[[class Name ::] [Function Name] \] Symbol Name

```
[[Class Name] [\Function Name] \] Symbol Name
```

When a symbol is specified using the module and function names, the symbol is searched. However, when only the symbol name is specified, the search is made as follows:

1.Local symbols within function in Current Scope
2.The class member which can access with the this pointer (when C++)
3.Static symbols within module in Current Scope
4.Global symbols

If a global symbol has the same name as a local symbol in the Current Scope, specify "\" or "::" at the start of global symbol. By doing so, you can explicitly show that is a global symbol.

An automatic variable can be referred to only when the variable is in memory. Otherwise, specifying an automatic variable causes an error.

1.17.2 Referring to C/C++ Variables

C/C++ variables can be specified using the same descriptions as in the source program written in C/C++.

■ Specifying C/C++ Variables

C/C++ variables can be specified using the same descriptions as in the source program. The address of C/ C++ variables should be preceded by the ampersand symbol "&". Some examples are shown in the Table 1.17-1 .

Table 1.17-1 Examples of Specifying Variables

Example of Variables		Example of Specifying Variables	Semantics
Regular Variable	int data;	data	Value of data
Pointer	char *p;	*p	Value pointed to by p
Array	char a[5];	a[1]	Value of second element of a

Structure	struct stag {
	char c;
	int i;
	};
	struct stag st;
	struct stag *stp;
		st.c
		stp- >c
			Value of member c of st
			Value of member c of the structure to which stp points

Union	union utag {
	char c;
	int i;
	} uni;
		uni.i	Value of member i of uni

Address of variable	int data;	&data	Address of data

Reference type	int i;
	int &ri = i;
		ri	Same as i

Class	class X {
	static int i;
	} cls;
	int X::i;
		cls.i
		X::i
			Value of member i of class X
			Same as cls.i

Member pointer class	class X {
	short cs;
	} clo;
	short X::* ps = &X::cs;
	X* clp = &clo;
		clo.*ps
		clp- >*ps
			Same as clo.cs
			Same as clp- >cs

■ Notes on C/C++ Symbols

The C/C++ compiler outputs symbol information with "_" prefixed to global symbols. For example, the symbol main outputs symbol information _main. However, SOFTUNE Workbench permits access using the symbol name described in the source to make program debugging described in C/C++ easier.

Consequently, a symbol name described in C/C++ and a symbol name described in assembler, which should both be unique, may be identical.

In such a case, the symbol name in the Current Scope normally is preferred. To refer to a symbol name outside the Current Scope, specify the symbol with the module name.

If there are duplicated symbols outside the Current Scope, the symbol name searched first becomes valid. To refer to another one, specify the symbol with the module name.

CHAPTER 2
Dependence Function

This chapter describes the functions dependent on F^2MC-8L/8FX family MCU.

2.1 Simulator Debugger

This section describes the functions of the simulator debugger for the F^{2}MC-8L/8FX family.

■ Simulator Debugger

The simulator debugger (later referred as simulator) simulates the MCU operations (executing instructions, memory space, I/O ports, interrupts, reset, etc.) with software to evaluate a program.

It is used to evaluate an uncompleted system, the operation of single units, etc.

■ Simulation Range

The simulator simulates the MCU operations (instruction operations, memory space, interrupts, reset, power-save consumption mode, etc.) with software. Peripheral I/Os, such as a timer, DMAC and serial I/O, other than the CPU core of the actual chip are not supported as peripheral resources. I/O space to which peripheral I/Os are connected is treated as memory space. There is a method for simulating interrupts like timer interrupts, and data input to memory like I/O ports. For details, see the sections concerning I/O port simulation and interrupt simulation.

- Instruction simulation
- Memory simulation
- I/O port simulation (Input port)
- I/O port simulation (Output port)
- Interrupt simulation
- Reset simulation
- Power-save mode simulation

2.1.1 Instruction Simulation

This section describes the instruction simulation executed by SOFTUNE Workbench.

■ Instruction Simulation

This simulates the operations of all instructions supported by the F^2MC-8L/8FX. It also simulates the changes in memory and register values due to such instructions.

2.1.2 Memory Simulation

This section describes the memory simulation executed by SOFTUNE Workbench.

■ Memory Simulation

The simulator must first secure memory space to simulate instructions because it simulates the memory space secured in the host machine memory.

One of the following operations is required.

- To secure the memory area, either use the [Setup] - [Memory Map] command, or the Set Map command in the Command window.
- Load the file output by the Linkage Editor (Load Module File) using either the [Debug] - [Load target file] command, or the LOAD/OBJECT command in the Command window.

■ Simulation Memory Space

Memory space access attributes can be specified byte-by-byte using the [Setup] - [Memory Map] command. The access attribute of unspecified memory space is Undefined.

■ Memory Area Access Attributes

Access attributes for memory area can be specified as shown in Table 2.1-1 . A guarded access break occurs if access is attempted against such access attribute while executing a program. When access is made by a program command, such access is allowed regardless of the attribute, CODE, READ or WRITE. However, access to memory in an undefined area causes an error.

Table 2.1-1 Types of Access Attributes

Attribute	Semantics
CODE	Instruction operation enabled
READ	Data read enabled
WRITE	Data write enabled
undefined	Attribute undefined (access prohibited)

2.1.3 I/O Port Simulation

This section describes I/O port simulation executed by SOFTUNE Workbench.

■ I/O Port Simulation (Input Port)

There are two types of simulations in I/O port simulation: input port simulation, and output port simulation. Input port simulation has the following types:

- Whenever a program reads the specified port, data is input from the pre-defined data input source.
- Whenever the instruction execution cycle count exceeds the specified cycle count, data is input to the port.

To set an input port, use the [Setup] - [Debug Environment] - [I/O Port] command, or the Set Inport command in the Command window.

Up to 4096 port addresses can be specified for the input port. The data input source can be a file or a terminal. After reading the last data from the file, the data is read again from the beginning of the file. If a terminal is specified, the input terminal is displayed at read access to the set port.

A text file created by an ordinary text editor, or a binary file containing direct code can be used as the data input source file. When using a text file, input the input data inside commas (,). When using a binary file, select the binary radio button in the input port dialog.

■ I/O Port Simulation (Output Port)

At output port simulation, whenever a program writes data to the specified port, writing is executed to the data output destination.

To set an output port, either use the [Setup] - [Debug Environment] - [I/O Port] command, or the Set Outport command in the Command window.

Up to 4096 port addresses can be set as output ports. Select either a file or terminal (Output Terminal window) as the data output destination.

An output destination file must be either a text file that can be referred to by regular editors, or a binary file. To output a binary file, select the Binary radio button in the Output Port dialog.

2.1.4 Interrupt Simulation

This section describes interrupt simulation executed by SOFTUNE Workbench.

■ Interrupt Simulation

Simulate the operation of the MCU in response to an interrupt request. The methods of generating interrupts are as follows:

- Execute instructions for the specified number of cycles while the program is running (during execution of executable commands) to generate interrupts corresponding to the specified interrupt numbers and cancel the interrupt generating conditions.
- Continue to generate interrupts each time the number of instruction execution cycles exceeds the specified number of cycles.

The method of generating interrupts is set by the [Setup]-[Debug environment]-[Interrupt] command. If interrupts are masked by the interrupt enable flag when the interrupt generating conditions are established, the interrupts are generated after they are unmasked.

MCU operation in response to an interrupt request is also supported for the following exception handling:

- Execution of undefined instructions
- Address error in program access
  (Program access to internal RAM area and internal I/O area)

2.1.5 Reset Simulation

This section describes the reset simulation executed by SOFTUNE Workbench.

■ Reset Simulation

The simulator simulates the operation when a reset signal is input to the MCU using the [Debug]-[Run]-[Reset MCU] command and initializes the registers. The function for performing reset processing by operation of MCU instructions (writing to RST bit in standby control register) is also supported. In this case, the reset message (Reset) is displayed on the status bar.

2.1.6 Power-Save Consumption Mode Simulation

This section describes the power-save consumption mode simulation executed by SOFTUNE Workbench.

■ Power-Save Consumption Mode Simulation

The MCU enters the power-save consumption mode in accordance with the MCU instruction operation (Write to SLEEP bit or STOP bit of standby control register). Once in the sleep mode or stop mode, a message ("sleep" for sleep mode, "stop" for stop mode) is displayed on the Status Bar. The loop keeps running until either an interrupt request is generated, or the [Run] - [Abort] command is executed. Each cycle of the loop increments the count by 1. During this period, I/O port processing can be operated. Writing to the standby control register using a command is not prohibited.

2.2 Emulator Debugger (MB2141)

This section describes the functions of the emulator debugger for the F^{2}MC-8L family.

■ Emulator Debugger

The emulator debugger (later referred as emulator) is a software to evaluate a program by controlling an ICE from a host computer via a communications line (RS-232C, LAN).

Before using this emulator, the ICE must be initialized.

For further details, refer to the SOFTUNE WORKBENCH Operation Manual Appendix B "Download Monitor Program", and Appendix C "Setting LAN Interface".

2.2.1 Setting Operating Environment

Before operating the emulator, set the operating environment such as the MCU operation mode, memory mapping, the timer minimum measurement unit, etc. However, each setting has a default. No setup is required if the defaults are used.

■ MCU Operation Modes

The operation mode setting varies when a regular MCU is used and when a F^2MC-8L piggy back/ evaluation chip is used.

● MCU mode

- Single chip mode (MCU Mode 0)
- External ROM mode (MCU Mode 1)
- Internal ROM mode with external access function (MCU Mode 2)

■ Memory Mapping

A memory space can be allocated to the user memory or the emulation memory. However, certain restrictions apply to the space that can be set, depending on the selected MCU mode.

■ Timer Minimum Measurement Unit

Select either 1 µs or 100 ns as the emulator timer minimum measurement unit for measuring time.

2.2.1.1 MCU Operation Mode

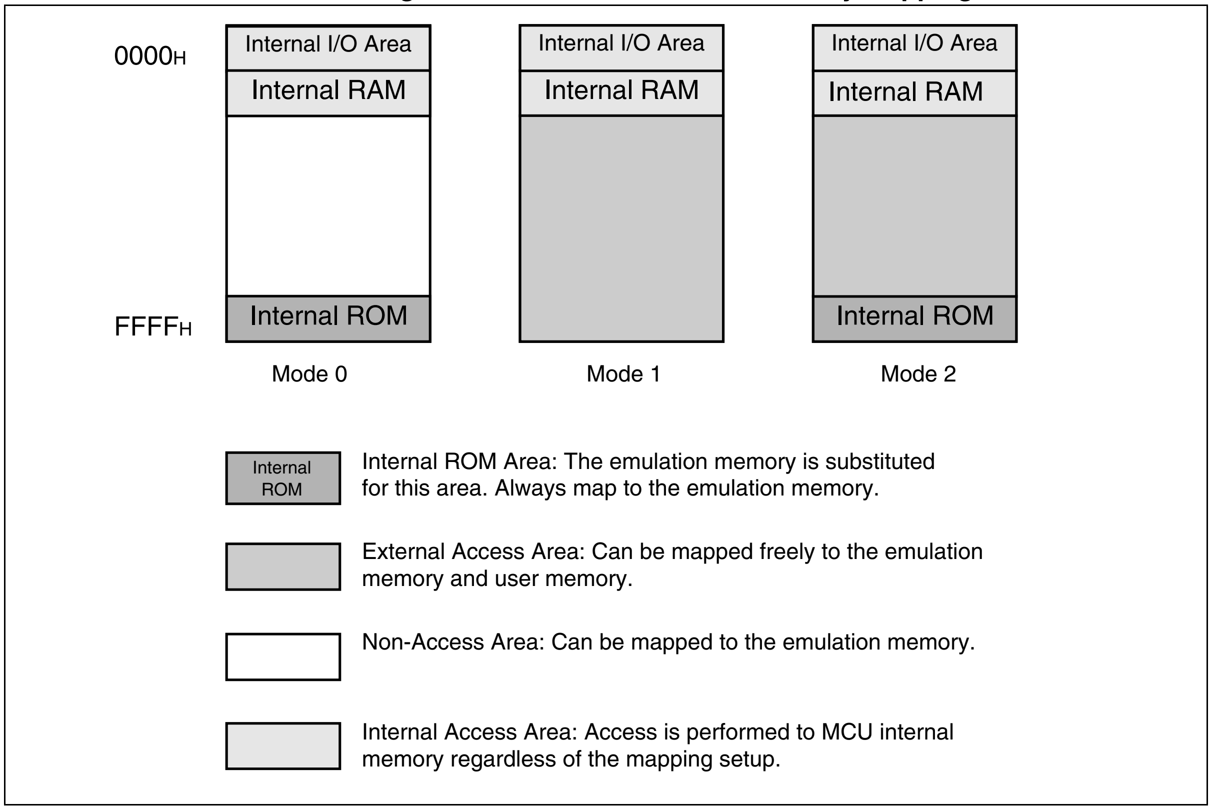
There are three MCU operation modes as follows:

- **Single chip mode (MCU Mode 0)**
- **External ROM mode (MCU Mode 1)**
- **Internal ROM mode with external access function (MCU Mode 2)**

■ Setting MCU Operation Mode

The MCU operation mode varies depending on the product type; refer to the Hardware Manual for each MCU for further details.

Figure 2.2-1 MCU Modes and Memory Mapping


0000H
FFFFH
Internal I/O Area
Internal RAM
Internal ROM
Mode 0
Internal I/O Area
Internal RAM
Mode 1
Internal I/O Area
Internal RAM
Internal ROM
Mode 2
Internal ROM
Internal ROM Area: The emulation memory is substituted for this area. Always map to the emulation memory.
External Access Area: Can be mapped freely to the emulation memory and user memory.
Non-Access Area: Can be mapped to the emulation memory.
Internal Access Area: Access is performed to MCU internal memory regardless of the mapping setup.

As shown in Figure 2.2-1 , memory mapping operation varies depending on MCU mode. Internal RAM area (internal RAM, internal register, and internal I/O) cannot map to the emulation memory because it accesses internal MCU regardless of mapping setup.

2.2.1.2 Memory Area Types

A unit to allocate memory is called an area. There are three different area types.

■ Memory Area Types

A unit to allocate memory is called an area. Up to 20 areas can be set in 1-byte units. There is no limit on the size of an area. An access attribute can be set for each area.

There are three different area types as follows:

● User Memory Area

Memory space in the user system is called the user memory area and this memory is called the user memory.

To set the user memory area, use the SET MAP command.

● Emulation Memory Area

Memory space substituted for emulator memory is called the emulation memory area, and this memory is called emulation memory.

The user system bus master (DMAC, etc.) cannot access emulation memory.

To set the emulation memory area, use the SET MAP command.

● Undefined Area

A memory area that does not belong to any of the areas described above is part of the user memory area. This area is specifically called the undefined area.
The undefined area can be set to either NOGUARD area, which can be accessed freely, or GUARD area, which cannot be accessed. Select either setup for the whole undefined area. If the area attribute is set to GUARD, a guarded access error occurs if access to this area is attempted.

2.2.1.3 Memory Mapping

Memory space can be allocated to the user memory and the emulation memory, etc., and the attributes of these areas can be specified.
However, the MCU internal resources are not dependent on this mapping setup and access is always made to the internal resources.

■ Access Attributes for Memory Areas

The access attributes shown in Table 2.2-1 can be specified for memory areas.

A guarded access break occurs if access is attempted in violation of these attributes while executing a program.

When access to the user memory area and the emulation memory area is made using program commands, such access is allowed regardless of the READ, WRITE attributes. However, access to memory with the GUARD attribute in the undefined area, causes an error.

Table 2.2-1 Types of Access Attributes

Area	Attribute	Description
User Memory	Read	Data Read and Instruction Execution Enabled
Emulation Memory	Write	Data Write Enabled
Undefined GUARD	GUARD	Access Disabled
	NOGUARD	No check of access attribute

When access is made to an area without the WRITE attribute by executing a program, a guarded access break occurs after the data has been rewritten if the access target is the user memory area. However, if the access target is the emulation memory area, the break occurs before rewriting. In other words, write-protection (memory data cannot be overwritten by writing) can be set for the emulation memory area by not specifying the WRITE attribute for the area.

This write-protection is only enabled for access made by executing a program, and is not applicable to access by commands.

■ Creating and Displaying Memory Map

Use the following commands for memory mapping.

- SET MAP: Sets memory map
- SHOW MAP: Displays memory map
- CANCEL MAP: Changes memory map setting to undefined

[Example]

```
>SET MAP /USER H'0..H'1FFF
>SET MAP /READ/EMULATION H'FF00..H'FFFF
>SET MAP/GUARD
>SHOW MAP
address          attribute          type
0000 .. 1FFF     code read write    user
FF00 .. FFFF     code read          emulation
-----------------------------------------------------------------
 undefined  area : guard
 setup possibility : user = 19          emulation=19
```

2.2.1.4 Timer Minimum Measurement Unit

The timer minimum measurement unit affects the sequencer, the emulation timer and the performance measurement timer.

■ Setting Timer Minimum Measurement Unit

Choose either 1 µs or 100 ns as the timer minimum measurement unit for the emulator of measuring time.

The minimum measurement unit for the following timers is changed depending on this setup.

- Timer values of sequencer (timer conditions at each level)
- Emulation timer
- Performance measurement timer

Table 2.2-2 shows the minimum measurement time length of each timer when 1 µs or 100 ns is selected as the minimum measurement unit.

When the minimum measurement unit is changed, the measurement values of each timer are cleared as well. The default setting is 1 µs.

Table 2.2-2 Minimum Measurement Time Length of Each Timer

	1 µs selected	100 ns selected
Sequencer timer	About 16 s	About 1.6 s
Emulation timer	About 70 min	About 7 min
Performance measurement timer	About 70 min	About 7 min

Use the following commands to control timers.

- SET TIMERSCALE command: Sets minimum measurement unit for timers
- SHOW TIMERSCALE command: Displays status of minimum measurement unit setting for timers

[Example]

```
>SET TIMERSCALE/100N
>SHOW TIMERSCALE
Timer scale : 100ns
>
```

2.2.2 On-the-fly Executable Commands

Certain commands can be executed even while executing a program. This is called "on-the-fly" execution.

■ On-the-fly Executable Commands

Certain commands can be executed on-the-fly. If an attempt is made to execute a command that cannot be executed on-the-fly, an error "MCU is busy" ossurs. Table 2.2-3 lists major on-the-fly executable functions. For further details, refer to the SOFTUNE WORKBENCH Command Reference Manual.

Meanwhile, on-the-fly execution is enabled only when executing the MCU from the menu or the tool button. On-the-fly commands cannot be executed when executing the GO command, etc., from the command window.

Table 2.2-3 Major Functions Executable in On-the-fly Mode

Function	Restrictions	Major Commands
MCU reset	-	RESET
Displaying MCU execution status	-	SHOW STATUS
Displaying trace data	Enabled only when trace function disabled	SHOW TRACE SHOW MULTITRACE
Enable/Disable trace	-	ENABLE TRACE DISABLE TRACE
Displaying execution time measurement value (Timer)	-	SHOW TIMER
Memory operation (Read/Write)	Emulation memory only operable Read only enabled in mirror area	ENTER EXAMINE COMPARE FILL MOVE DUMP SEARCH MEMORY SHOW MEMORY SET MEMORY
Line assembly, Disassembly	Emulation memory only enabled Mirror area, Disassembly only enabled	ASSEMBLE DISASSEMBLE
Load, Save program	Emulation memory only enabled Mirror area, save only enabled	LOAD SAVE

2.2.3 On-the-fly Memory Access

While on-the-fly, the area mapped to the emulation memory is Read/Write enabled, but the area mapped to the user memory is Read-only enabled.

■ Read/Write Memory while On-the-fly

The user memory cannot be accessed while on-the-fly (executing MCU). However, the emulation memory can be accessed. (The cycle-steal algorithm eliminates any negative effect on the MCU speed.)

This emulator allows the user to use part of the emulation memory as a mirror area. The mirror area holds a copy of the user memory. Using this mirror area makes the user memory to Read-only enabled function available while on-the fly.

However, at least one time access must be allowed before the emulation memory with the mirror area setting has the same data as the user memory. The following copy types allow the emulation memory with the mirror area setting to have the same data as the user memory.

- Copying only required portion using memory access commands

  Data in the specified portion can be copied by executing a command that accesses memory. The following commands access memory.

- Memory operation commands

  SET MEMORY, SHOW MEMORY, EXAMINE, ENTER, COMPARE, FILL, MOVE, SEARCH MEMORY, DUMP, COPY, VERIFY

- Data load/save commands

  LOAD, SAVE

Figure 2.2-2 Access to Mirror Area while MCU Suspended

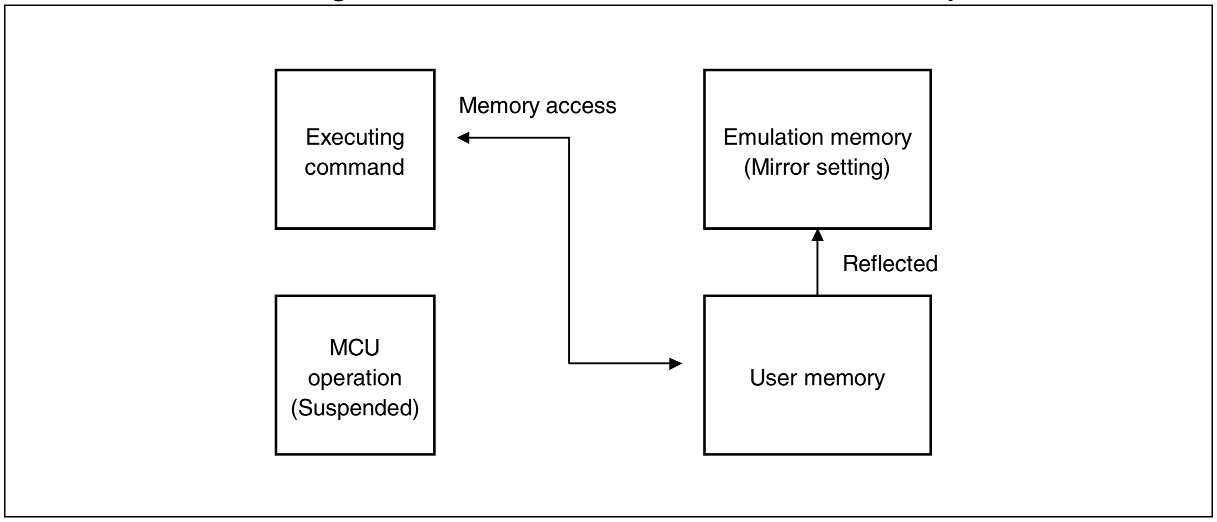
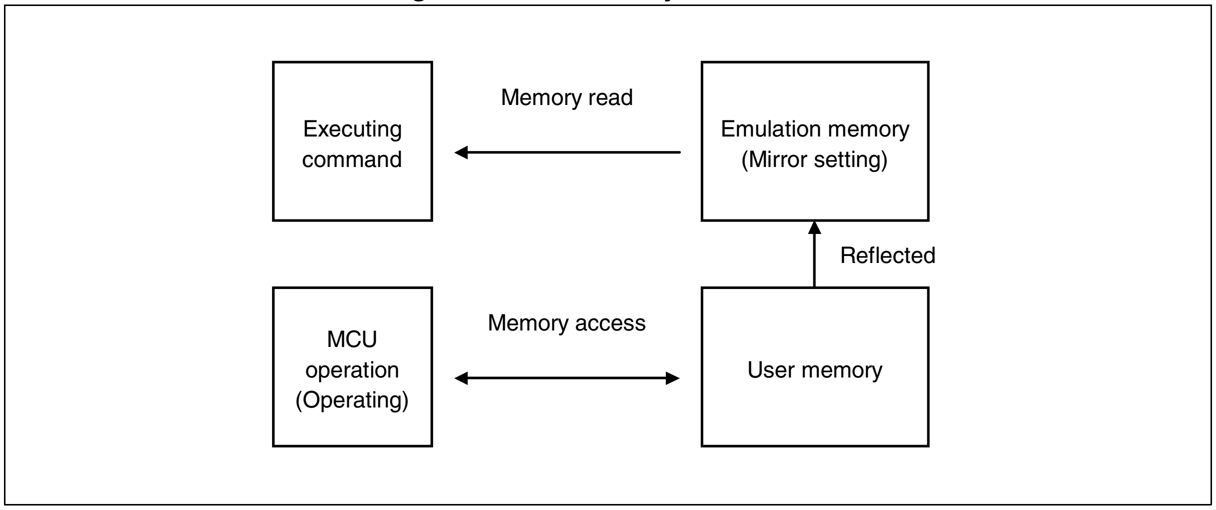


Figure 2.2-3 On-the-fly Access to Mirror Area



Note:

Memory access by a bus master other than the MCU is not reflected in the mirror area.

2.2.4 Events

The emulator can monitor the MCU bus operation, and generate a trigger at a specified condition called an event.
In this emulator, event triggers are used in order to determine which function event triggers are used accounting to event modes for the following functions;
- **Sequencer**
- **Sampling condition for multi-trace**
- **Measuring point in performance measurement**

■ Setting Events

Up to eight events can be set.

Table 2.2-4 shows the conditions that can be set for events.

Table 2.2-4 Conditions for Setting Events

Condition	Description
Address	Memory location (Address bit masking enabled)
Data	8-bit data (data bit masking enable) NOT specified enable
Status	Select from among data read, data write, instruction execution and data modify.
External probe	8-bit data (bit masking enable)

Notes:

- In instruction execution, an event trigger is generated only when an instruction is executed. This status cannot be specified concurrently with other status.
- The data modify is a function to generate the event trigger when the data of specified address is rewritten. When the data modify is specified in the status, the specified data is ignored. This status cannot be specified concurrently with other status.

Use the following commands to set an event.

SET EVENT:	Sets event
SHOW EVENT:	Display event setup status
CANCEL EVENT:	Deletes event
ENABLE EVENT:	Enable event
DISABLE EVENT:	Disable event

[Example]

```
>SET EVENT  1,func1
>SET EVENT/WRITE 2,data[2],!d=h'10
>SET EVENT/MODIFY 3,102
```

Reference:

An event can be set in the Event window as well.

■ Event Modes

There are three event modes as listed below. To determine which function event triggers are used for, select one using the SET MODE command. The default is normal mode.

The event value setting are made for each mode, so switching the event mode changes the event settings as well.

- **Normal Mode**

  Event triggers are used for sequencer.

  Since the sequencer can perform control at 8 levels, it can control sequential breaks, time measurement and trace sampling. Real-time tracing in the normal mode is performed by single trace (tracing function that samples program execution continuously).

- **Multitrace Mode**

  Event triggers are used for multitracing (trace function that samples data before and after event trigger occurrence).

- **Performance Mode**

  Event triggers are used for performance measurement to measure time duration between two event trigger occurrences and count of event trigger occurrences.

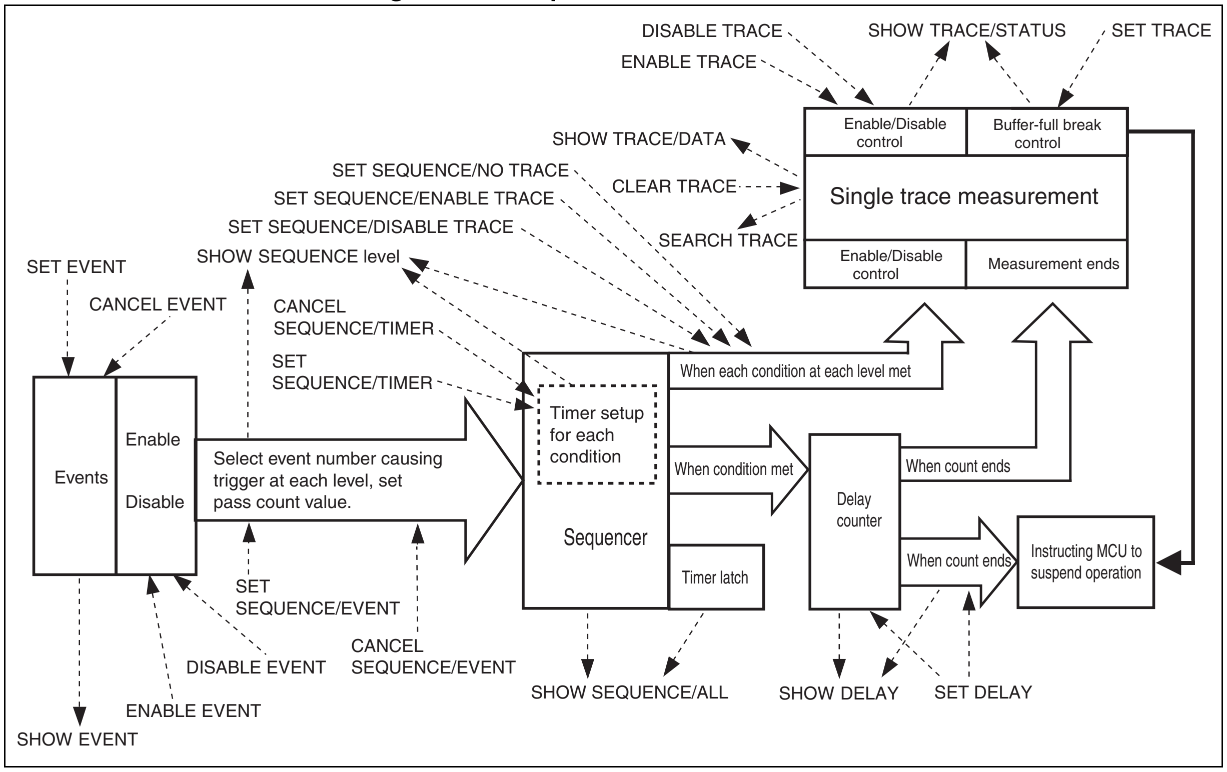
2.2.4.1 Operation in Normal Mode

As shown in the figure below, the event trigger set in the normal mode performs input to the sequencer. In the sequencer, either branching to any level, or terminating the sequencer, can be specified as an operation at event trigger occurrence. This enables debugging (breaks, limiting trace, measuring time) while monitoring program flow.

■ Operation in Normal Mode

The termination of sequencer triggers the delay counter. When the delay counter reaches the specified count, sampling for the single trace terminates. A break normally occurs at this point, but if necessary, the program can be allowed to run on without a break.

Figure 2.2-4 Operation in Normal Mode



■ Event-related Commands in Normal Mode

Since the real-time trace function in the normal mode is actually the single trace function, the commands can be used to control.

Table 2.2-5 shows the event-related commands that can be used in the normal mode.

Table 2.2-5 Event-related Commands in Normal Mode

Mode	Usable Command	Function
Normal Mode	Set Event Show Event Cancel Event Enable Event Disable Event	Sets event Displays event setup status Delete event Enables event Disables event
	Set Sequence Show Sequence Cancel Sequence Enable Sequence Disable Sequence	Sets sequencer Displays sequencer setup status Cancels sequencer Enables sequencer Disables sequencer
	Set Delay Show Delay	Sets delay count Displays delay count setup status
	Set Trace Show Trace Search Trace Enable Trace Disable Trace Clear Trace	Sets trace buffer-full break Displays trace data Searches trace data Enables trace function Disables trace function Clears trace data

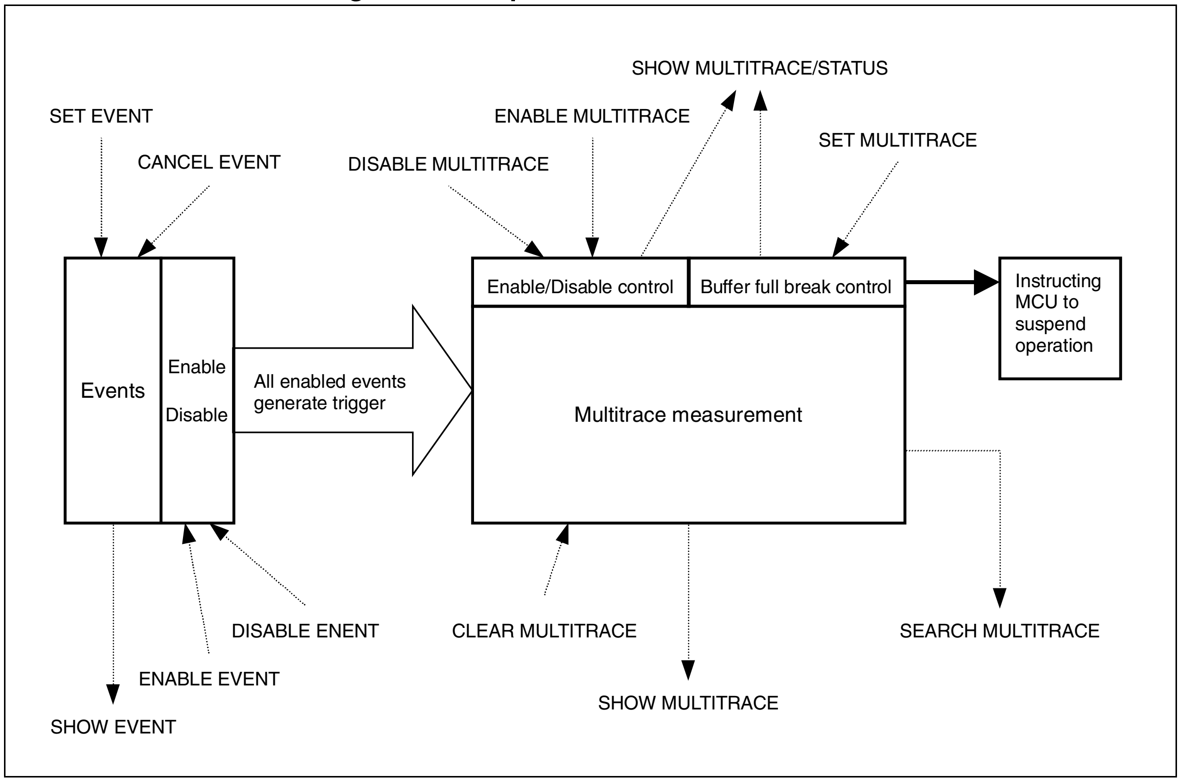
2.2.4.2 Operation in Multitrace Mode

When the multitrace mode is selected as the event mode, the real-time trace function becomes the multitrace function, and events are used as triggers for multitracing.

■ Operation in Multitrace Mode

Multitracing is a trace function that samples data before and after an event trigger occurrence. When the multitrace mode is selected as the event mode, the real-time trace function becomes the multitrace function, and events are used as triggers for multitracing.

Figure 2.2-5 Operation in Multitrace Mode



■ Event-related Commands in Multitrace Mode

Table 2.2-6 shows the event-related commands that can be used in the multirace mode.

Table 2.2-6 Event-related Commands in Multitrace Mode

Mode	Usable Command	Function
Multitrace Mode	Set Event Show Event Cancel Event Enable Event Disable Event	Sets event Displays event setup status Deletes event Enables event Disables event
	Set MultiTrace Show MultiTrace Search MultiTrace Enable MultiTrace Disable MultiTrace Clear MultiTrace	Sets trace buffer-full break Displays trace data Searches trace data Enables trace function Disables trace function Clears trace data

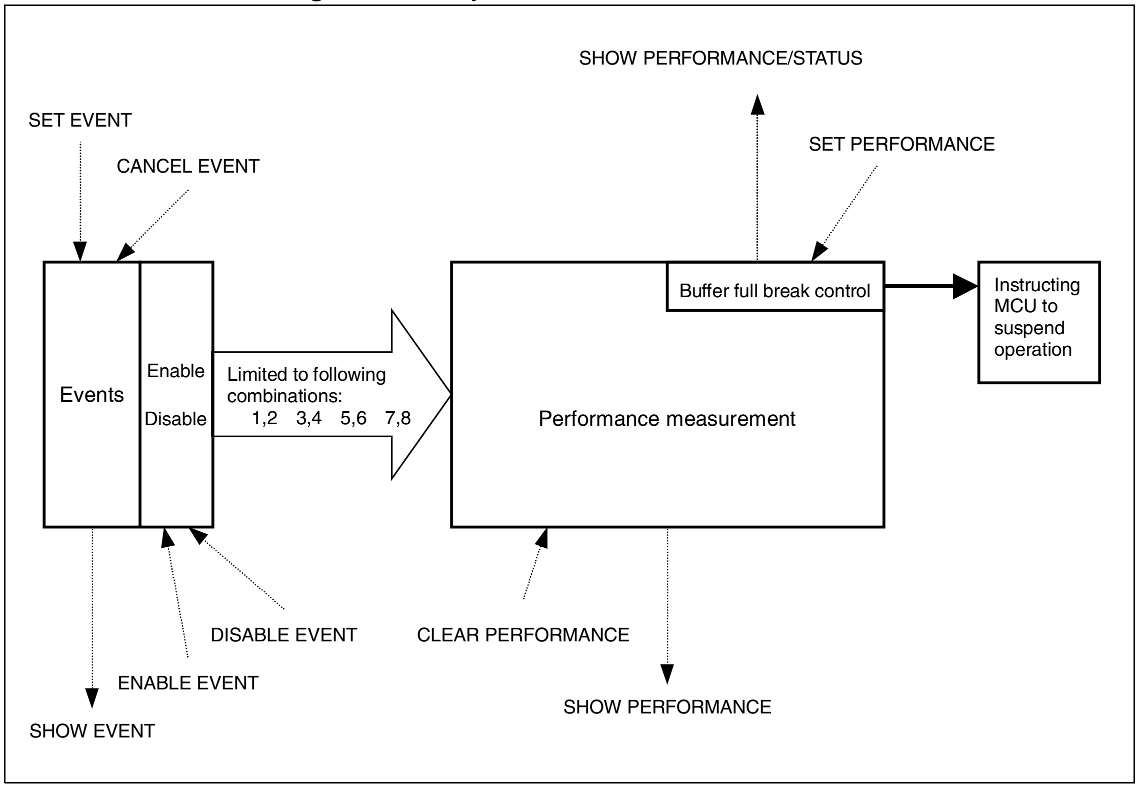
2.2.4.3 Operation in Performance Mode

Event triggers set in the performance mode are used to measure performance. The time duration between two event occurrences can be measured and the event occurrences can be counted.

■ Operation in Performance Mode

The event triggers that are set in the performance mode are used to measure performance. The time duration between two event occurrences can be measured and the event occurrences can be counted.

Figure 2.2-6 Operation in Performance Mode



■ Event-related Commands in Performance Mode

Table 2.2-7 shows the event-related commands that can be used in the performance mode.

Table 2.2-7 Event-related Commands in Performance Mode

Mode	Usable Command	Function
Performance Mode	Set Event Show Event Cancel Event Enable Event Disable Event	Sets event Displays event setup status Deletes event Enables event Disables event
	Set Performance Show Performance Clear Performance	Sets performance Displays performance setup status Clears performance measurement data

2.2.5 Control by Sequencer

This emulator has a sequencer to control events. By using this sequencer, sampling of breaks, time measurement and tracing can be controlled while monitoring program flow (sequence). A break caused by this function is called a sequential break.
To use this function, set the event mode to normal mode using the SET MODE command. Use the SET EVENT command to set events.

■ Control by Sequencer

As shown in Table 2.2-8 , controls can be made at 8 different levels.

At each level, 8 events and 1 timer condition (9 conditions in total) can be set.

A timer condition is met when the timer count starts at entering a given level and the specified time is reached.

For each condition, the next operation can be specified when the condition is met. Select any one of the following.

- Move to required level.
- Terminate sequencer.

The conditions set for each level are determined by OR. Therefore, if any one condition is met, the sequencer either moves to the required level, or terminates. In addition, trace sampling suspend/resume can be controlled when a condition is met.

Table 2.2-8 Sequencer Specifications

Function	Specifications
Level count	8 levels
Conditions settable for each level	8 event conditions (1 to 16777216 times pass count can be specified for each condition.) 1 timer condition (Up to 16 s. in 1µs unit or up to 1.6 s. in 100 ns units can be specified.*)
Operation when condition met	Branches to required level or terminates sequencer. Controls trace sampling.
Other function	Timer latch enable at level branching
Operation when sequencer terminates	Starts delay counter.

*: The minimum measurement unit for Timer value can be set to either 1 µs or 100 ns using the SET TIMERSCALE command.

2.2.5.1 Setting Sequencer

The sequencer operates in the following order:

1. The sequencer starts from level 1 simultaneously with the start of program executing.

2. Depending on the setting at each level, branching to the required level is performed when the condition is met.

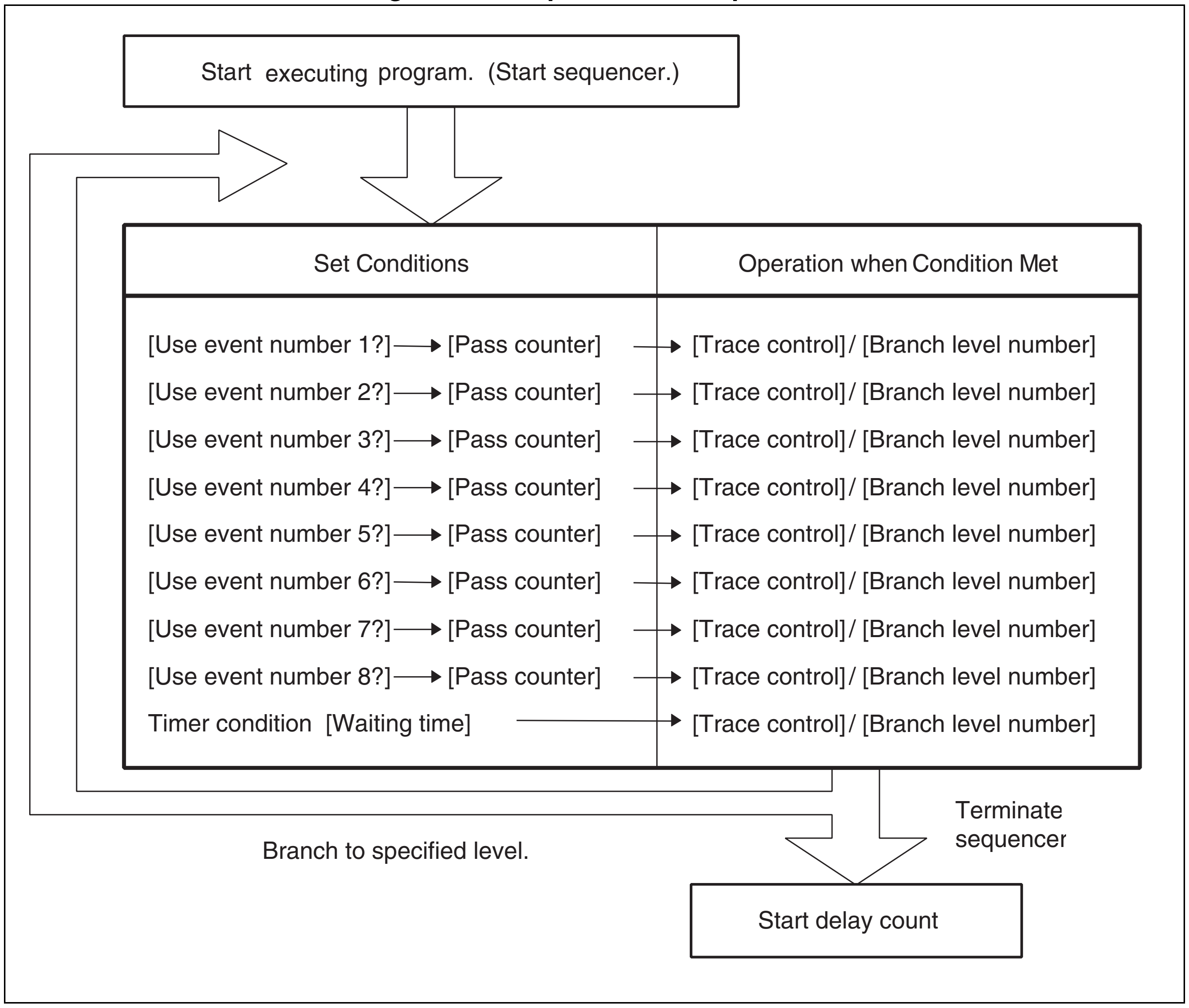
3. When sequencer termination is specified, the sequencer terminates when the condition is met.

4. When the sequencer terminates, the delay counter starts counting.

■ Setting Sequencer

Figure 2.2-7 shows the sequencer operation.

Figure 2.2-7 Operation of Sequencer


Start executing program. (Start sequencer.)
Set Conditions
Operation when Condition Met
[Use event number 1?] → [Pass counter] → [Trace control] / [Branch level number]
[Use event number 2?] → [Pass counter] → [Trace control] / [Branch level number]
[Use event number 3?] → [Pass counter] → [Trace control] / [Branch level number]
[Use event number 4?] → [Pass counter] → [Trace control] / [Branch level number]
[Use event number 5?] → [Pass counter] → [Trace control] / [Branch level number]
[Use event number 6?] → [Pass counter] → [Trace control] / [Branch level number]
[Use event number 7?] → [Pass counter] → [Trace control] / [Branch level number]
[Use event number 8?] → [Pass counter] → [Trace control] / [Branch level number]
Timer condition [Waiting time] → [Trace control] / [Branch level number]
Branch to specified level.
Terminate sequencer
Start delay count

[Setup Examples]

- Terminate sequencer when event 1 occurs.

  >SET SEQUENCE/EVENT 1,1,J=0
- Terminate sequencer when event 2 occurs 16 times.

  >SET SEQUENCE/EVENT 1,2,16,J=0
- Terminate sequencer when event 2 occurs after event 1 occurred. However, do not terminate sequencer if event 3 occurs between event 1 and event 2.

  >SET SEQUENCE/EVENT 1,1,J=2

  >SET SEQUENCE/EVENT 2,2,J=0

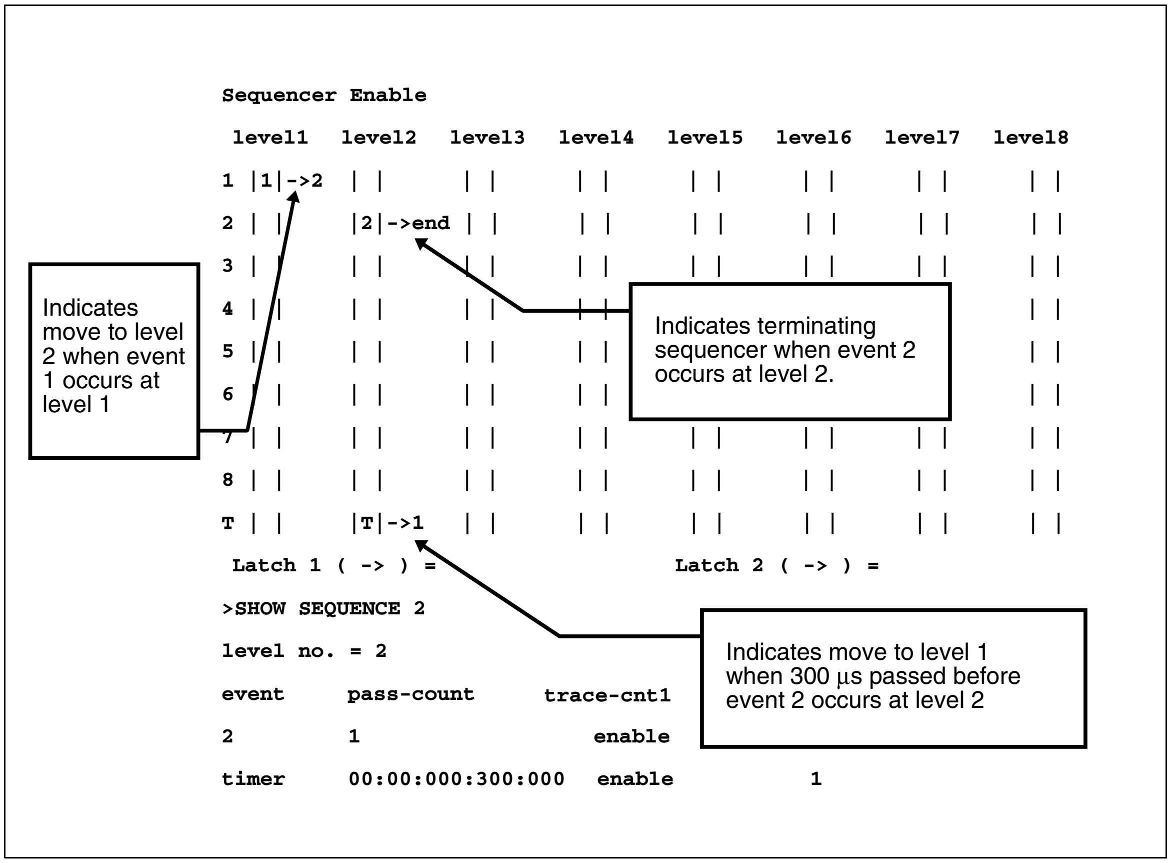
  >SET SEQUENCE/EVENT 2,3,J=1
- Terminate sequencer when event 2 occurs less than 300 µs after event 1 occurred.

  >SET SEQUENCE/EVENT 1,1,J=2

  >SET SEQUENCE/EVENT 2,2,J=0

  >SET SEQUENCE/TIMER 2,300,J=1

  >SHOW SEQUENCE



```
Sequencer Enable
 level1   level2   level3   level4   level5   level6   level7   level8
1 |1|->2  | |      | |      | |      | |      | |      | |      | |
2 | |     |2|->end | |      | |      | |      | |      | |      | |
3 | |     | |      | |      | |      | |      | |      | |      | |
4 | |     | |      | |      | |                                 | |
5 | |     | |      | |      | |                                 | |
6 | |     | |      | |      | |                                 | |
7 | |     | |      | |      | |      | |      | |      | |      | |
8 | |     | |      | |      | |      | |      | |      | |      | |
T | |     |T|->1   | |      | |      | |      | |      | |      | |
 Latch 1 ( -> ) =                    Latch 2 ( -> ) =
>SHOW SEQUENCE 2
level no. = 2
event      pass-count      trace-cnt1
2          1                    enable
timer      00:00:000:300:000   enable           1
```

Indicates move to level 2 when event 1 occurs at level 1
Indicates terminating sequencer when event 2 occurs at level 2.
Indicates move to level 1 when 300 µs passed before event 2 occurs at level 2

2.2.5.2 Break by Sequencer

A program can suspend program execution when the sequencer terminates. This break is called a sequential break.

■ Break by Sequencer

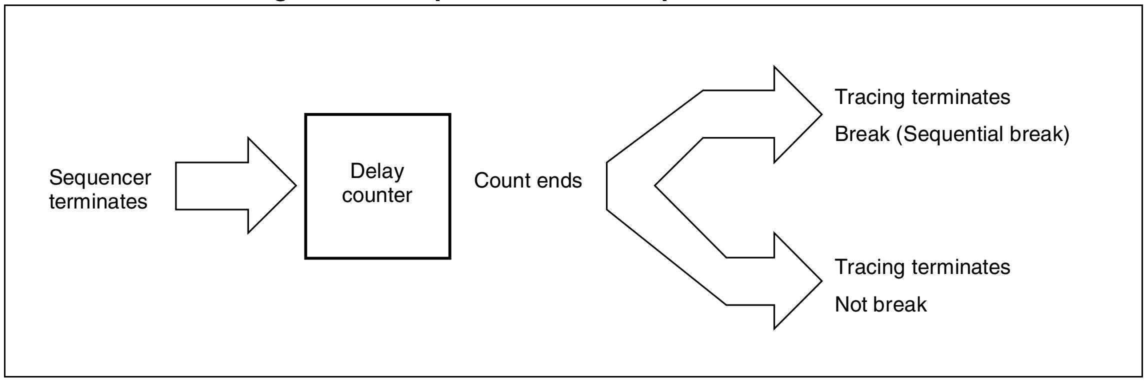
A program can suspend program execution when the sequencer terminates. This break is called a sequential break.

As shown in Figure 2.2-8 , the delay count starts when the sequencer terminates, and after delay count ends, either "break" or "not break but tracing only terminates" is selected as the next operation.

To make a break immediately after the sequencer terminates, set delay count to 0 and specify "Break after delay count terminates". Use the SET DELAY command to set the delay count and the operation after the delay count.

The default is delay count 0, and Break after delay count.

Figure 2.2-8 Operation when sequencer terminates



[Examples of Delay Count Setups]

- Break when sequencer terminates.

  ```
  >SET DELAY/BREAK 0
  ```

- Break when 100-bus-cycle tracing done after sequencer terminates.

  ```
  >SET DELAY/BREAK 100
  ```

- Terminate tracing, but do not break when sequencer terminates.

  ```
  >SET DELAY/NOBREAK 0
  ```

- Terminate tracing, but do not break when 100-bus-cycle tracing done after sequencer terminates.

  ```
  >SET DELAY/NOBREAK 100
  ```

2.2.5.3 Trace Sampling Control by Sequencer

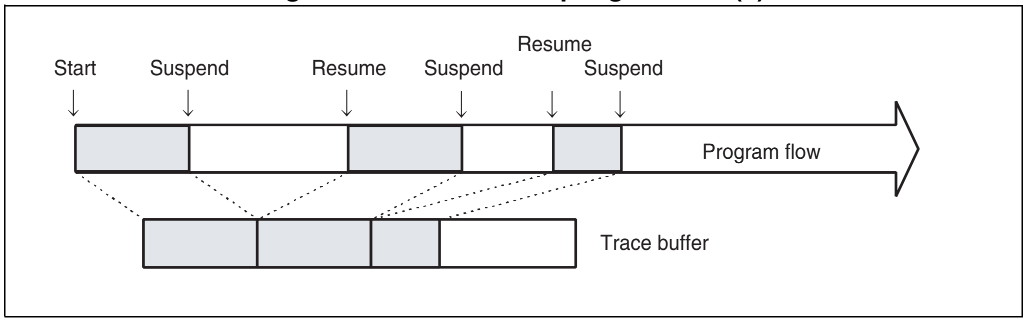
When the event mode is in the normal mode, real-time trace executing tracing called single trace.
If the trace function is enabled, single trace samples all the data from the start of executing a program until the program is suspended.

■ Trace Sampling Control by Sequencer

Sets up suspend/resume trace sampling for each condition at each level of the sequencer. Figure 2.2-9 shows the trace sampling flow.

For example, it is possible to suspend trace sampling when event 1 occurs, and then resume trace sampling when event 2 occurs. Sampling trace data can be restricted.

Figure 2.2-9 Trace Sampling Control (1)

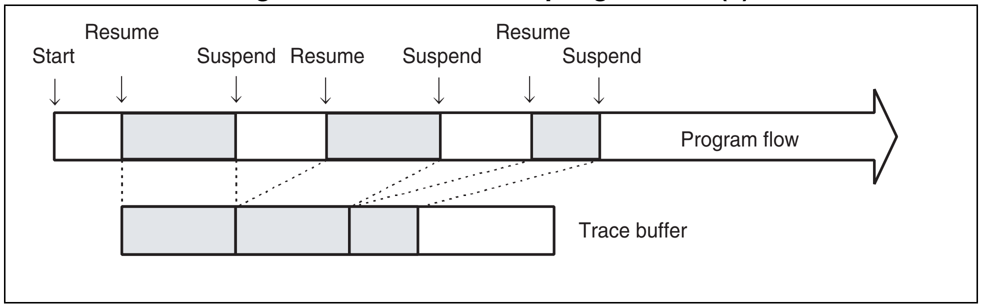


As shown in Figure 2.2-10 , trace sampling can be disabled during the period from the start of a program execution until the first condition occurs. For this setup, use the GO command or the SET GO command.

[Example]

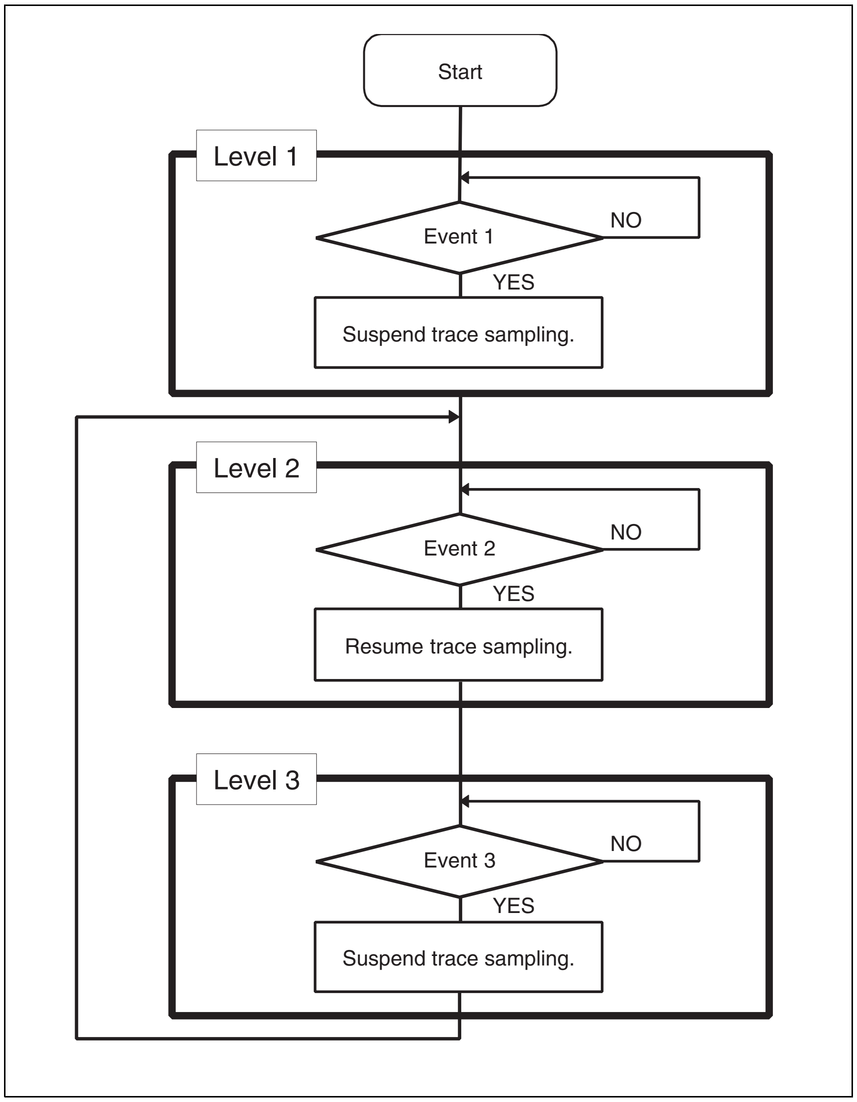
```
>GO/DISABLETRACE
>SET GO/DISABLETRACE
>GO
```

Figure 2.2-10 Trace Sampling Control (2)



[Setup Example]

Suspend trace sampling when event 1 occurs, and then resume at event 2 and keep sampling data until event 3 occurs.



```
>SET SEQUENCE/EVENT/DISABLETRACE 1,1,J=2
>SET SEQUENCE/EVENT/ENABLETRACE 2,2,J=3
>SET SEQUENCE/EVENT/DISABLETRACE 3,3,J=2
```

2.2.5.4 Time Measurement by Sequencer

Time can be measured using the sequencer. A time measurement timer called the emulation timer is used for this purpose. When branching is made from a specified level to another specified level, a timer value is fetched. Up to two emulation timer values can be fetched. This function is called the timer latch function.

■ Time Measurement by Sequencer

The time duration between two given points in a complex program flow can be measured using the timer latch function.

The timing for the timer latch can be set using the SET SEQUENCE command; the latched timer values can be diplayed using the SHOW SEQUENCE command.

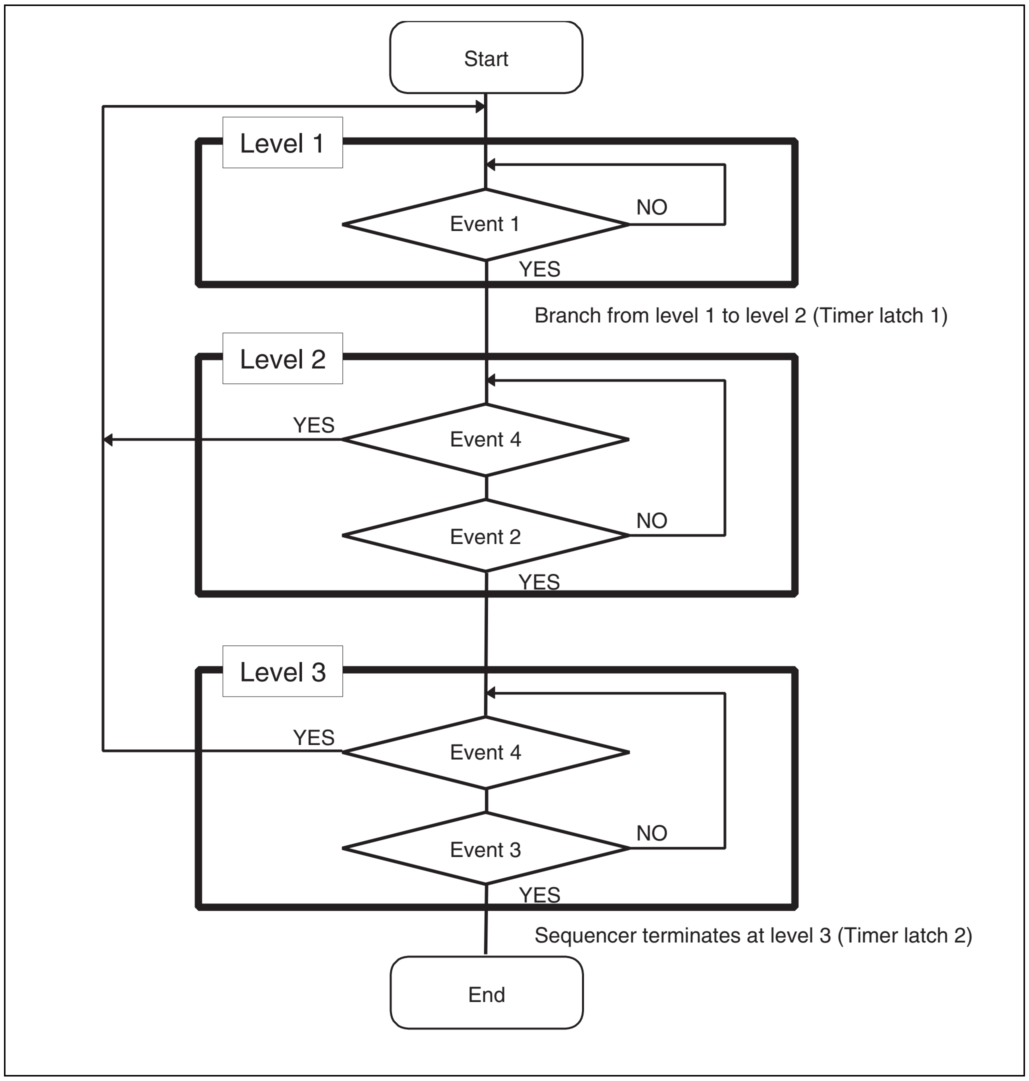
When a program starts execution, the emulation timer is initialized and then starts counting. Select either 1µs or 100 ns as the minimum measurement unit for the emulation timer. Set the measurement unit using the SET TIMESCALE command.

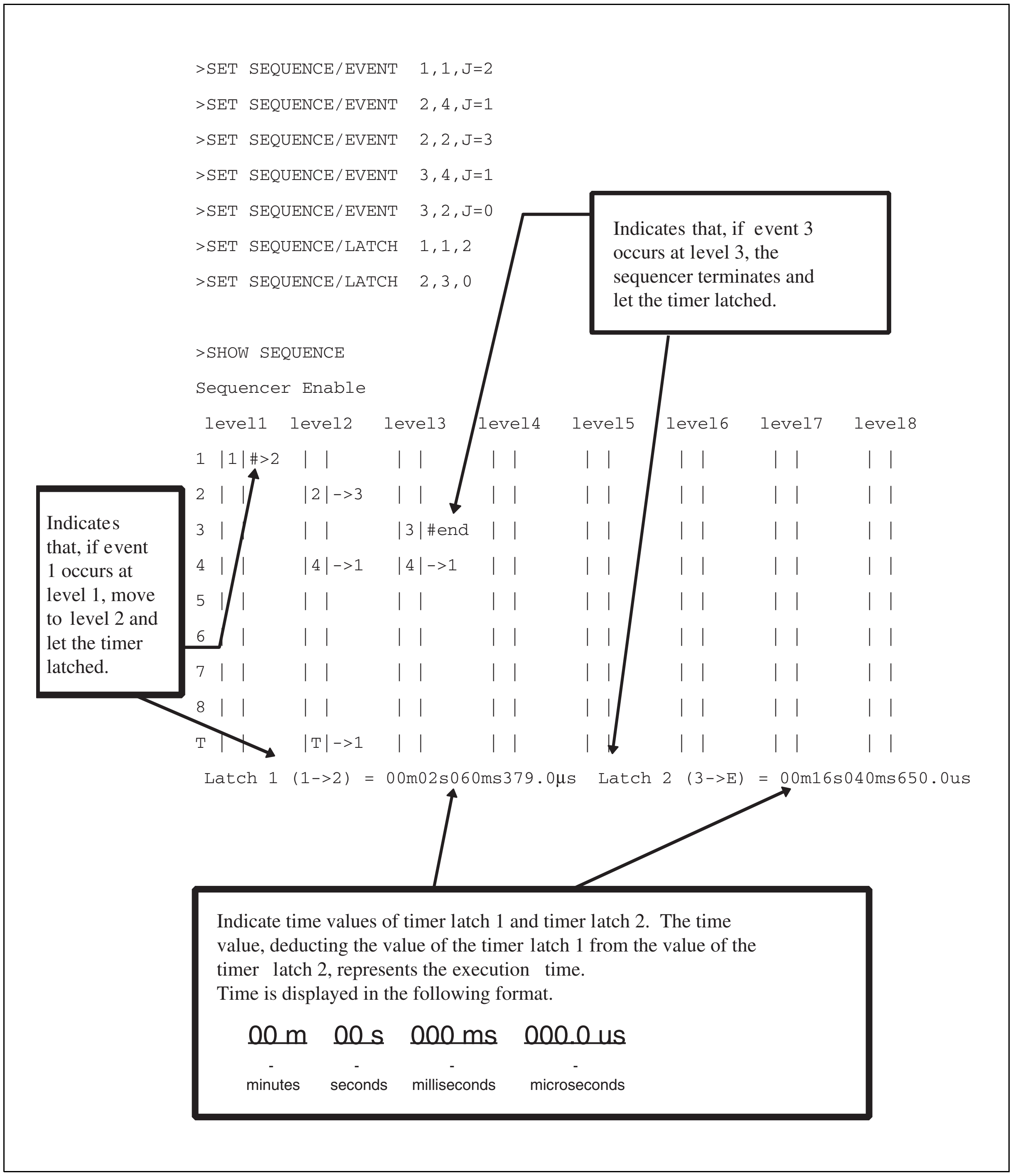
When 1 µs is selected, the maximum measured time is about 70 minutes; when 100 ns is selected, the maximum measured time is about 7 minutes. If the timer overflows during measurement, a warning message is displayed when the timer value is displayed using the SHOW SEQUENCE command.

2.2.5.5 Sample Flow of Time Measurement by Sequencer

In the following sample, when events are executed in the order of Event 1, Event 2 and Event 3, the execution time from the Event 1 to the Event 3 is measured. However, no measurement is made if Event 4 occurs anywhere between Event 1 and Event 3.

■ Sample Flow of Time Measurement by Sequencer


Start
Level 1
Event 1
NO
YES
Branch from level 1 to level 2 (Timer latch 1)
Level 2
YES
Event 4
Event 2
NO
YES
Level 3
YES
Event 4
Event 3
NO
YES
Sequencer terminates at level 3 (Timer latch 2)
End



```
>SET SEQUENCE/EVENT  1,1,J=2
>SET SEQUENCE/EVENT  2,4,J=1
>SET SEQUENCE/EVENT  2,2,J=3
>SET SEQUENCE/EVENT  3,4,J=1
>SET SEQUENCE/EVENT  3,2,J=0
>SET SEQUENCE/LATCH  1,1,2
>SET SEQUENCE/LATCH  2,3,0

>SHOW SEQUENCE
Sequencer Enable
 level1   level2   level3   level4   level5   level6   level7   level8
1 |1|#>2  | |      | |      | |      | |      | |      | |      | |
2 | |     |2|->3   | |      | |      | |      | |      | |      | |
3 | |     | |      |3|#end  | |      | |      | |      | |      | |
4 | |     |4|->1   |4|->1   | |      | |      | |      | |      | |
5 | |     | |      | |      | |      | |      | |      | |      | |
6 | |     | |      | |      | |      | |      | |      | |      | |
7 | |     | |      | |      | |      | |      | |      | |      | |
8 | |     | |      | |      | |      | |      | |      | |      | |
T | |     |T|->1   | |      | |      | |      | |      | |      | |
 Latch 1 (1->2) = 00m02s060ms379.0µs  Latch 2 (3->E) = 00m16s040ms650.0us
```

Indicates that, if event 3 occurs at level 3, the sequencer terminates and let the timer latched.

Indicates that, if event 1 occurs at level 1, move to level 2 and let the timer latched.

Indicate time values of timer latch 1 and timer latch 2. The time value, deducting the value of the timer latch 1 from the value of the timer latch 2, represents the execution time.
Time is displayed in the following format.

<u>00 m</u>	<u>00 s</u>	<u>000 ms</u>	<u>000.0 us</u>
-	-	-	-
minutes	seconds	milliseconds	microseconds

2.2.6 Real-time Trace

While execution a program, the address, data and status information, and the data sampled by an external probe can be sampled in machine cycle units and stored in the trace buffer. This function is called real-time trace.
In-depth analysis of a program execution history can be performed using the data recorded by real-time trace.
There are two types of trace sampling: single trace, which traces from the start of executing the program until the program is suspended, and multitrace, which starts tracing when an event occurs.

■ Trace Buffer

The data recorded by sampling in machine cycle units, is called a frame.

The trace buffer can store 32K frames (32768). Since the trace buffer has a ring structure, when it becomes full, it automatically returns to the start to overwrite existing data.

■ Trace Data

Data sampled by the trace function is called trace data.

The following data is sampled:

- Address
- Data
- Status Information
    - Access status: Read/Write/Internal access, etc.
    - Device status: Instruction execution, Reset, Hold, etc.
- External probe data
- Sequencer execution level

2.2.6.1 Single Trace

The single trace function traces all data from the start of executing a program until the program is aborted.

■ Function of Single Trace

The single trace is enabled by setting the event mode to normal mode using the SET MODE command.

The single trace function traces all data from the start of executing a program until the program is suspended.

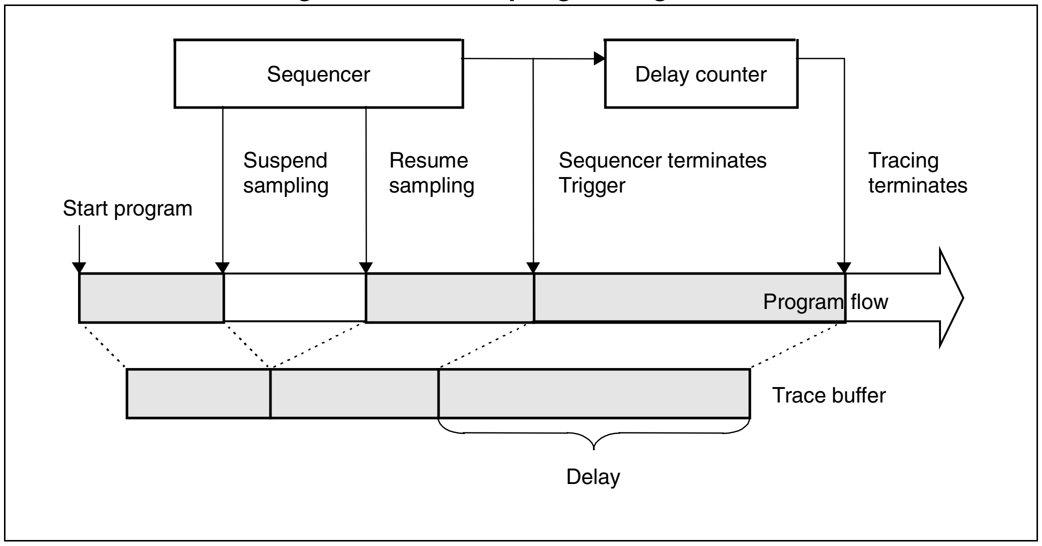
If the real-time trace function is enabled, data sampling continues execution to record the data in the trace buffer while the GO, STEP, CALL commands are being executed.

As shown in Figure 2.2-11 , suspend/resume trace sampling can be controlled by the event sequencer. Since the delay can be set between the sequencer terminating the trigger and the end of tracing, the program flow after an given event occurrence can be traced. The delay count is counted in bus cycle units, so it matches the sampled trace data count. However, nothing can be sampled during the delay count if trace sampling is suspended when the sequencer is terminated.

After the delay count ends, a break occurs normally due to the sequential break, but tracing can be terminated without a break.

Furthermore, a program can be allowed to break when the trace buffer becomes full. This break is called a trace-buffer-full break.

Figure 2.2-11 Sampling in Single Trace



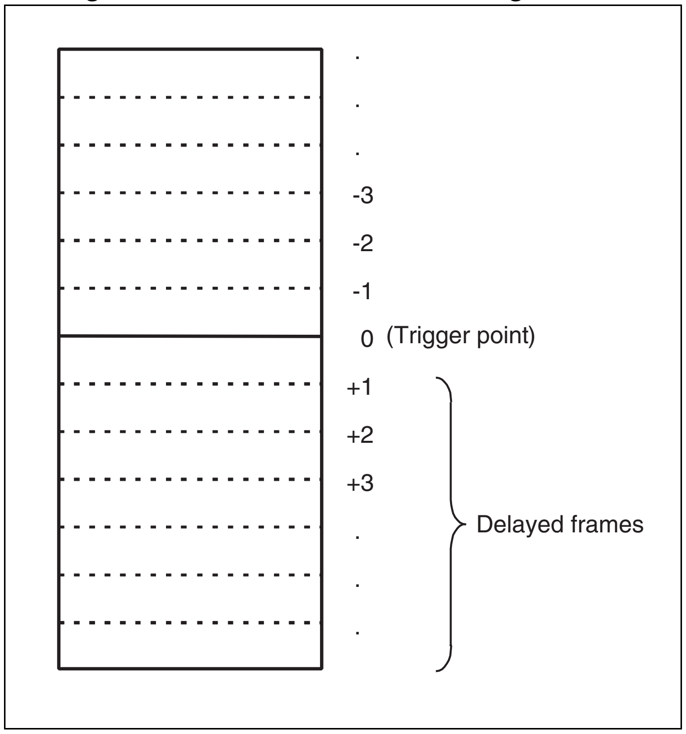
■ Frame Number and Step Number in Single Trace

The sampled trace data is numbered in frame units. This number is called the frame number.

When displaying trace data, the starting location in the trace buffer can be specified using the frame number. The trace data at the point where the sequencer termination trigger occurs is numbered 0; trace data sampled before reaching the trigger point is numbered negatively, and the data sampled after the trigger point is numbered positively (See Figure 2.2-12).

If there is no sequencer termination trigger point available, the trace data sampled last is numbered 0.

Figure 2.2-12 Frame Number in Single Trace



2.2.6.2 Setting Single Trace

The following settings 1 to 4 are required before executing single trace. Once these settings have been made, trace data is sampled when a program is executed.

1. Set event mode to normal mode.

2. Enable trace function.

3. Set events, sequencer, and delay count.

4. Set trace-buffer-full break.

■ Setting Single Trace

The following settings are required before executing single trace. Once these settings have been made, trace data is sampled when a program is executed.

(1) Set event mode to normal mode.

Use SET MODE command to make this setting.

(2) Enable trace function.

Use the ENABLE TRACE command. To disable the function, use the DISABLE TRACE command. The default is Enable.

(3) Set events, sequencer, and delay count.

Trace sampling can be controlled by setting the sequencer for events. If this function is not needed, there is no need of this setting.

To set events, use the SET EVENT command. To set the sequencer, use the SET SEQUENCE command.

Furthermore, set the delay count between sequencer termination and trace ending, and the break operation (Break or Not Break) when the delay count ends. If the data after event occurrence is not required, there is no need of this setting.

If Not Break is set, the trace terminates but no break occurs. To check trace data in on-the-fly, use this setup by executing the SET DELAY command.

Note:

When the sequencer termination causes a break (sequential break), the last executed machine cycle is not sampled.

(4) Set trace-buffer-full break.

The program can be allowed to break when the trace buffer becomes full. Use the SET TRACE command for this setting. The default is Not Break. Display the setup status using the SHOW TRACE/ STATUS command.

Table 2.2-9 lists trace-related commands that can be used in the single trace function.

Table 2.2-9 Trace-related Commands that can be used in the single trace function

Usable Command	Function
Set Event	Sets events
Show Event	Displays event setup status
Cancel Event	Deletes event
Enable Event	Enables event
Disable Event	Disables event
Set Sequence	Sets sequencer
Show Sequence	Displays sequencer setting status
Cancel Sequence	Cancels sequencer
Enable Sequence	Enables sequencer
Disable Sequence	Disables sequencer
Set Delay	Sets delay count value and operation after delay
Show Delay	Displays delay count setting status
Set Trace	Sets traces-buffer-full break
Show Trace	Displays trace data
Search Trace	Searchs trace data
Enable Trace	Enables trace function
Disable Trace	Disables trace function
Clear Trace	Clears trace data

2.2.6.3 Multitrace

The multitrace function samples data where an event trigger occurs for 8 frames before and after the event trigger.

■ Multitrace Function

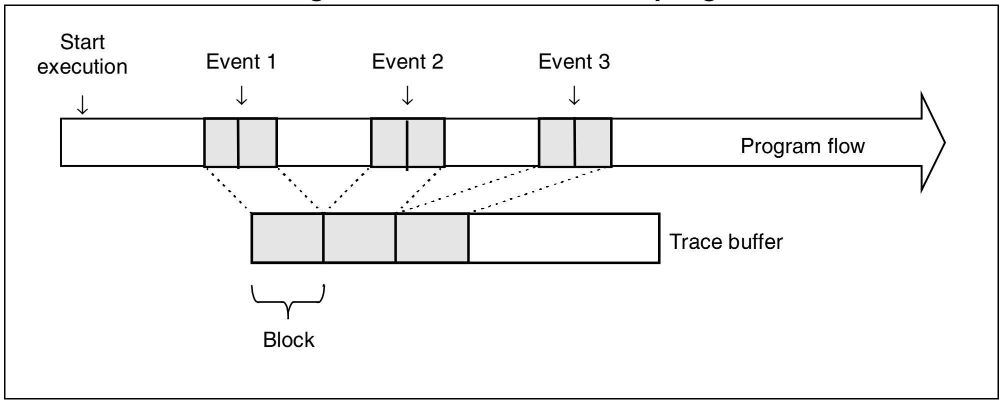
Execute multitrace by setting the event mode to the multitrace mode using the SET MODE command.

The multitrace function samples data where an event trigger occurs for 8 frames before and after the event trigger.

It can be used for tracing required only when a certain variable access occurs, instead of continuous tracing.

The trace data sampled at one event trigger (16 frames) is called a block. Since the trace buffer can hold 32K frames, up to 2048 blocks can be sampled. Multitrace sampling terminates when the trace buffer becomes full. At this point, a executing program can be allowed to break if necessary.

Figure 2.2-13 Multitrace Sampling



■ Multitrace Frame Number

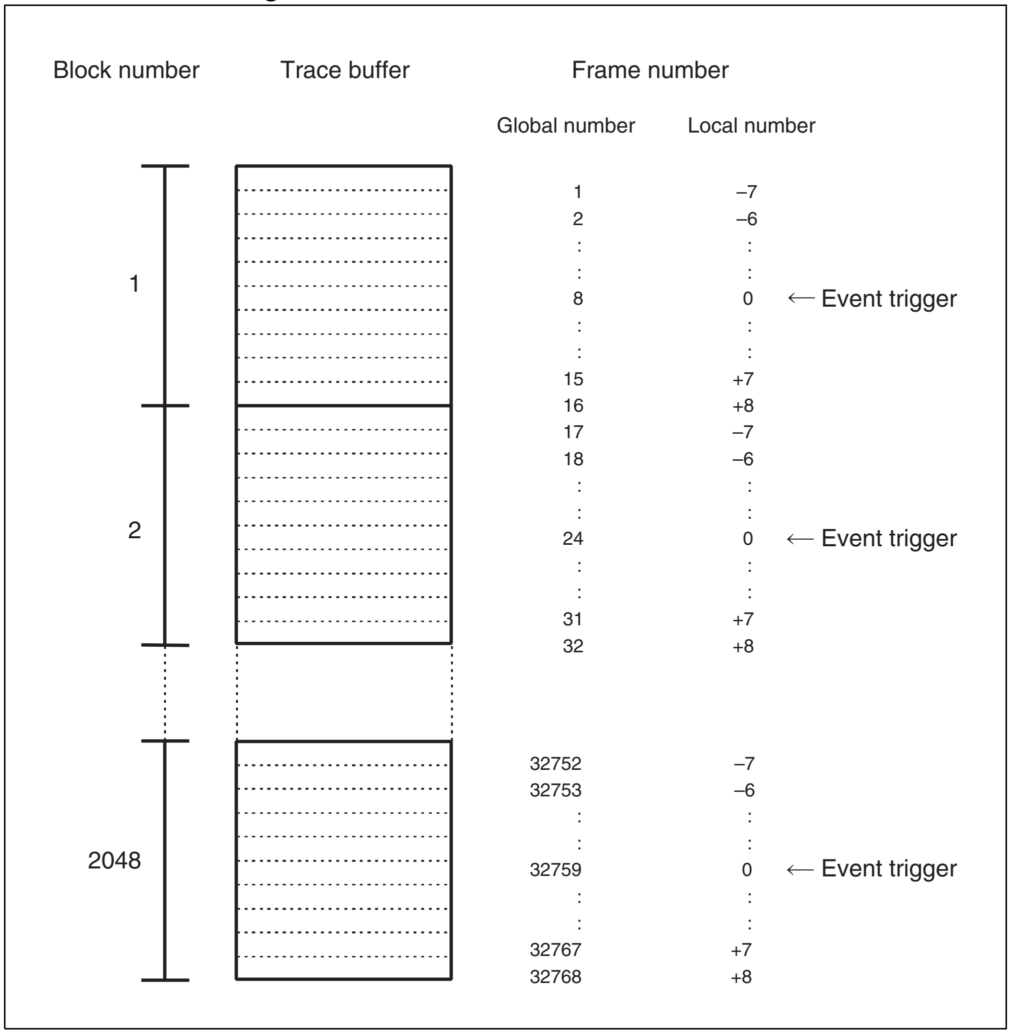
Sixteen frames of data are sampled each time an event occurs. This data unit is called a block, and each sampled block is numbered starting from 0. This is called the block number.

A block is a collection of 8 frames of sampled data before and after the event trigger occurs. At the event trigger point is 0, trace data sampled before reaching the event trigger point is numbered negatively, and trace data sampled after the event trigger point is numbered positively. These frame numbers are called local numbers (See Figure 2.2-14).

In addition to this local number, there is another set of frame numbers starting with the oldest data in the trace buffer. This is called the global number. Since the trace buffer can hold 32K frames, frames are numbered 1 to 32758 (See Figure 2.2-14).

To specify which frame data is displayed, use the global number or block and local numbers.

Figure 2.2-14 Frame Number in Multitrace



2.2.6.4 Setting Multitrace

Before executing the multitrace function, the following settings must be made. After these settings, trace data is sampled when a program is executed.
1. Set event mode to multitrace mode.
2. Enable trace function.
3. Set event.
4. Set trace-buffer-full break.

■ Setting Multitrace

Before executing the multitrace function, the following settings must be made. After these settings, trace data is sampled when a program is executed.

(1) Set event mode to multitrace mode.

Use the SET MODE command for this setting.

(2) Enable trace function.

Use the ENABLE MULTITRACE command. To disable the function, use the DISABLE MULTITRACE command.

(3) Set event.

Set an event that sampling. Use the SET EVENT command for this setting.

(4) Set trace-buffer-full break.

To break when the trace buffer becomes full, set the trace-buffer-full break. Use the SET MULTITRACE command for this setting.

Table 2.2-10 shows the list of trace-related commands that can be used in multitrace mode.

Table 2.2-10 Trace-related Commands that can be used in multitrace mode

Usable Command	Function
Set Event	Sets events
Show Event	Displays event setup status
Cancel Event	Deletes event
Enable Event	Enables event
Disable Event	Disables event
Set Multitrace	Sets trace-buffer-full break
Show Multitrace	Displays trace data
Search Multitrace	Searches trace data
Enable Multitrace	Enables multitrace
Disable Multitrace	Disables multitrace
Clear Multitrace	Clears trace data

2.2.6.5 Displaying Trace Data Storage Status

It is possible to display how much trace data is stored in the trace buffer. This status data can be read by specifying /STATUS to the SHOW TRACE command in the single trace mode and to the SHOW MULTITRACE command in the multitrace mode.

■ Displaying Trace Data Storage Status

It is possible to display how much trace data is stored in the trace buffer. This status data can be read by specifying/STATUS to the SHOW TRACE command in the single trace mode and to the SHOW MULTITRACE command in the multitrace mode.

Frame numbers displayed in the multitrace mode is the grobal number.

[Example]

- In Single Trace

```
 >SHOW TRACE/STATUS
en/dis      = enable              Trace function enabled
buffer full = nobreak             Buffer full break function disabled
sampling    = end                 Trace sampling terminates
flame no. = -00120 to  00050      Frame -120 to 50 store data
step no.  = -00091 to  00022      Step -91 to 22 store data
 >
```

- In Multitrace

```
 >SHOW MULTITRACE/STATUS
en/dis      = enable              Multitrace function enabled
buffer full = nobreak             Buffer full break function disabled
sampling    = end                 Trace sampling terminates
block no. = 1 to 5                Block 1 to 5 store data
frame no. = 00001 to 00159        Frame 1 to 159 store data
                                  (Global number)
```

2.2.6.6 Specify Displaying Trace Data Position

It is possible to specify from which data in the trace buffer to display. To do so, specify a frame number with the SHOW TRACE command in the single trace mode, or specify either a global number or a block number and local number with the SHOW MULTITRACE command in the multitrace mode. A range can also be specified.

■ Specify Displaying Trace Data Position

It is possible to specify from which data in the trace buffer to displays. To do this, specify a step or frame number with the SHOW TRACE command in the single trace, and specify either a global number or a block number and local number with the SHOW MULTITRACE command in the multitrace mode. A range can also be specified.

[Example]

- In Single Trace Mode

```
>SHOW TRACE -6                     Start displaying from frame -6
>SHOW TRACE -6..10                 Display from frame -6 to frame 10
```

- In Multitrace

```
>SHOW MULTITRACE/GLOBAL 500  Start displaying from frame 500 (Global number)
>SHOW MULTITRACE/LOCAL 2     Displaying block number 2
>SHOW MULTITRACE/LOCAL 2,-5..5Display from frame -5 to frame 5 of block number 2
```

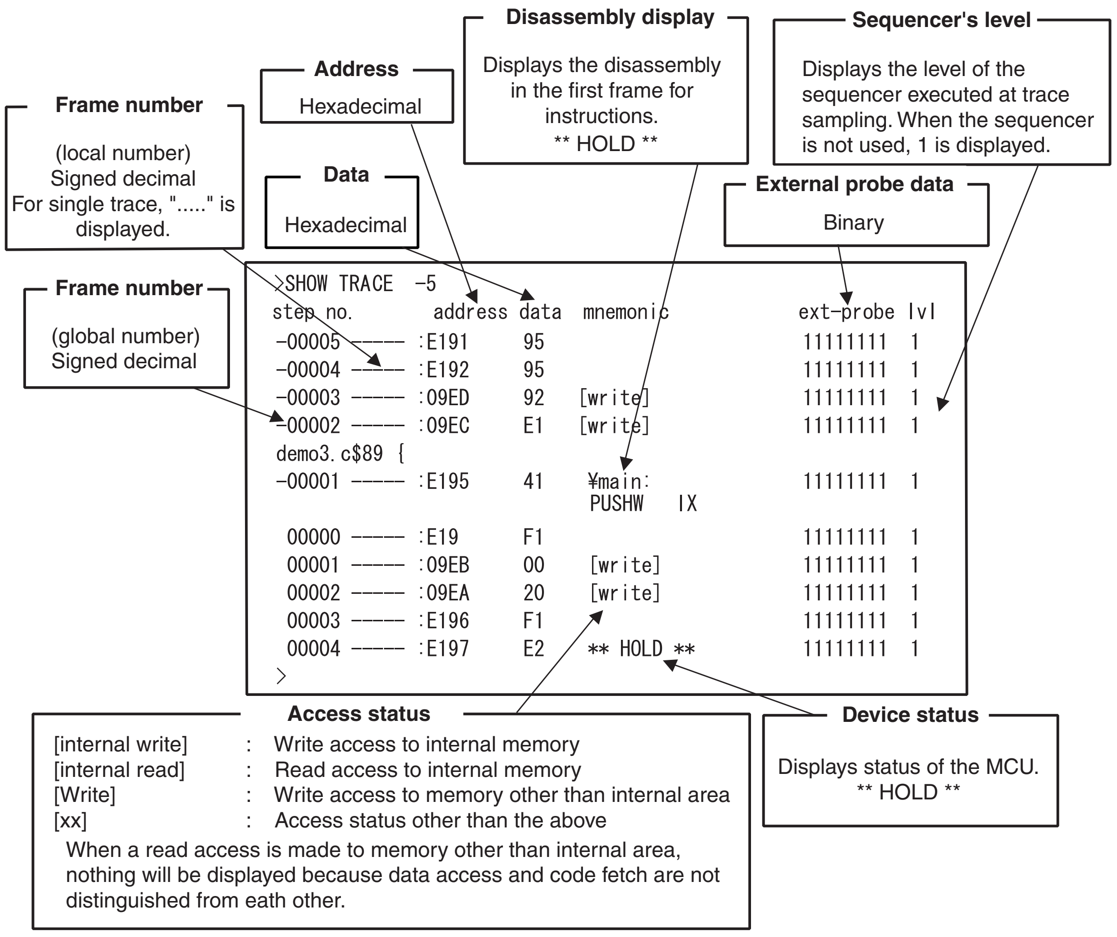
2.2.6.7 Display Format of Trace Data

A display format can be chosen by specifying a command identifier with the SHOW TRACE command in the single trace, and with the SHOW MULTITRACE command in the multitrace. The source line is also displayed if "Add source line" is selected using the SET SOURCE command.
There are three formats to display trace data:

- Display in all bus cycles	**(Specify /CYCLE.)**
- Display in only instruction execution	**(Specify /INSTRUCTION.)**
- Display in source line units	**(Specify /SOURCE.)**

■ Display All Bus Cycles (Specify /CYCLE.)

In this mode, data can be displayed in the following format.


Frame number
(local number)
Signed decimal
For single trace, "....." is displayed.
Frame number
(global number)
Signed decimal
Address
Hexadecimal
Data
Hexadecimal
Disassembly display
Displays the disassembly in the first frame for instructions.
** HOLD **
Sequencer's level
Displays the level of the sequencer executed at trace sampling. When the sequencer is not used, 1 is displayed.
External probe data
Binary
Device status
Displays status of the MCU.
** HOLD **
Access status

```
>SHOW TRACE  -5
step no.        address data  mnemonic              ext-probe lvl
-00005 ----- :E191     95                           11111111  1
-00004 ----- :E192     95                           11111111  1
-00003 ----- :09ED     92    [write]                11111111  1
-00002 ----- :09EC     E1    [write]                11111111  1
demo3.c$89 {
-00001 ----- :E195     41     ¥main:                11111111  1
                              PUSHW   IX
 00000 ----- :E19      F1                           11111111  1
 00001 ----- :09EB     00     [write]               11111111  1
 00002 ----- :09EA     20     [write]               11111111  1
 00003 ----- :E196     F1                           11111111  1
 00004 ----- :E197     E2     ** HOLD **            11111111  1
>
```

[internal write]	:	Write access to internal memory
[internal read]	:	Read access to internal memory
[Write]	:	Write access to memory other than internal area
[xx]	:	Access status other than the above

When a read access is made to memory other than internal area, nothing will be displayed because data access and code fetch are not distinguished from eath other.

For multi-trace, "Block number = XXXXXX" is displayed at the beginning of a block.

■ Display in Only Instruction Execution (Specify /INSTRUCTION.)

Only instruction execution is displayed in the same format as when /CYCLE is specified.

Data, access status, and device status are not displayed.

[Example]

```
>SHOW TRACE/INSTRUCTION -5
frame no.      address        mnemonic              ext-probe   lvl
demo3.c$89 {
-00001      ----- :E195       \main:                11111111    1
                              PUSHW     IX
 00003      ----- :E196       MOVW      A,SP        11111111    1
 00005      ----- :E197       MOVW      IX,A        11111111    1
 00007      ----- :E198       PUSHW     A           11111111    1
 00011      ----- :E199       PUSHW     A           11111111    1
 00015      ----- :E19A       PUSHW     A           11111111    1
demo3.c$91            int cc = 1;
 00019      ----- :E19B       MOVW      A,#0001     11111111    1
 00022      ----- :E19E       MOVW      @IX-02,A    11111111    1
demo3.c$93            numdt = 10;
 00027      ----- :E1A0       MOVW      A,#000A     11111111    1
>
```

■ Display in Source Line Units (Specify /SOURCE.)

Only the source line can be displayed.

[Example]

```
 >SHOW TRACE/SOURCE -5
frame no.          source
-00001:    -----   :demo3.c$89 {
 00019:    -----   :            demo3.c$91     int cc  =  1;
 00027:    -----   :            demo3.c$93     numdt = 10;
 00035:    -----   :            demo3.c$94     ackdat = 0;
 00041:    -----   :            demo3.c$96     sort (&datpl);        /* data sorting */
 00054:    -----   ;            demo3.c$147    struct st *dat;
 00071:    -----   :            demo3.c$152     ackdat += 5;
 00082:    -----   :            demo3.c$153     nckdat = ackdat;
 00086:    -----   :            demo3.c$154     for (j=0 ; j<numdt-1; j++) {
>
```

2.2.6.8 Reading Trace Data On-the-fly

Trace data can be read while executing a program. However, this is not possible during sampling. Disable the trace function or terminate tracing before attempting to read trace data.

■ Reading Trace Data On-the-fly in Single Trace

To disable the trace function, use the DISABLE TRACE command. Check whether or not the trace function is currently enabled by executing the SHOW TRACE command with /STATUS specified, or by using built-in variable, %TRCSTAT.

Tracing terminates when the delay count ends after the sequencer has terminated. If Not Break is specified here, tracing terminates without a break operation. It is possible to check whether or not tracing has terminated by executing the SHOW TRACE command with /STATUS specified, or by using built-in variable, %TRCSAMP.

To read trace data, use the SHOW TRACE command; to search trace data, use the SEARCH TRACE command. Use the SET DELAY command to set the delay count and break operation after the delay count.

[Example]

```
 >GO
 >>SHOW TRACE/STATUS
en/dis        = enable
buffer full   = nobreak
sampling      = on          <- Trace sampling continues.
 >>SHOW TRACE/STATUS
en/dis        = enable
buffer full   = nobreak
sampling      = end         <- Trace sampling ends.
frame no.     = -00805 to  00000
step no.      = -00262 to  00000
 >>SHOW TRACE -10
frame no.     address       data   mnemonic           ext-probe    lvl
00010 _ _ _ _: F000         04     MOV    A,#55       11111111     1
00009 _ _ _ _: F001         55                        11111111     1
00008 _ _ _ _: F002         45     MOV    60,A        11111111     1
00007 _ _ _ _: F003         60                        11111111     1
.               .           .
```

If the CLEAR TRACE command is executed with the trace ending state, trace data sampling can be re-executed by re-executing the sequencer from the beginning.

■ Reading Trace Data On-the-fly in the Multitrace

Use the DISABLE MULTITRACE command to disable the trace function before reading trace data. Check whether or not the trace function is currently enabled by executing the SHOW MULTITRACE command with /STATUS specified, or by using built-in variable %TRCSTAT.

To read trace data, use the SHOW MULTITRACE command; to search trace data, use the SEARCH MULTITRACE command.

[Example]

```
 >GO
>>SHOW MULTITRACE/STATUS
en/dis          = enable
buffer full     = nobreak
sampling        = on
 >>DISABLE MULTITRACE
 >>SHOW MULTITRACE/STATUS
en/dis          = disable
buffer full     = nobreak
sampling        = end
block no.       = 1 to 20
frame no.       = 00001 to 00639
 >>SHOW MULTITRACE  1
frame no.       address     data   mnemonic            ext-probe    lvl
block no. = 1
00001 -00007: F000          04     MOV    A,#55        11111111     1
00002 -00006: F001          55                         11111111     1
00003 -00005: F002          45     MOV    60,A         11111111     1
00004 -00004: F003          60                         11111111     1
                   .                      .                         .
                   .                      .                         .
```

2.2.7 Measuring Performance

It is possible to measure the time and pass count between two events. Repetitive measurement can be performed while executing a program in real-time, and when done, the data can be totaled and displayed.
Using this function enables the performance of a program to be measured. To measure performance, set the event mode to the performance mode using the SET MODE command.

■ Performance Measurement Function

The performance measurement function allows the time between two event occurrences to be measured and the number of event occurrences to be counted. Up to 32767 event occurrences can be measured.

- **Measuring Time**

  Measures time interval between two events.

  Events can be set at 8 points (1 to 8). However, in the performance measurement mode, the intervals, starting event number and ending event number are combined as follows. Four intervals have the following fixed event number combination:

Interval	Starting Event Number	Ending Event Number
1	1	2
2	3	4
3	5	6
4	7	8

- **Measuring Count**

  The specified events become performance measurement points automatically, and occurrences of that particular event are counted.

2.2.7.1 Performance Measurement Procedures

Performance can be measured by the following procedure:

- **Set event mode.**
- **Set minimum measurement unit for timer.**
- **Specify performance-buffer-full break.**
- **Set events.**
- **Execute program.**
- **Display measurement result.**
- **Clear measurement result.**

■ Setting Event Mode

Set the event mode to the performance mode using the SET MODE command. This enables the performance measurement function.

[Example]

```
>SET MODE/PERFORMANCE
>
```

■ Setting Minimum Measurement Unit for Timer

Using the SET TIMERSCALE command, choose either 1 µs or 100 ns as the minimum measurement unit for the timer used to measure performance. The default is 1 µs.

When the minimum measurement unit is changed, the performance measurement values are cleared.

[Example]

```
>SET TIMERSCALE/1U           <- Set 1 µs as minimum unit.
>
```

■ Setting Performance-Buffer-Full Break

When the buffer for storing performance measurement data becomes full, a executing program can be broken. This function is called the performance-buffer-full break. The performance buffer becomes full when an event occurs 32767 times.

If the performance-buffer-full break is not specified, the performance measurement ends, but the program does not break.

[Example]

```
>SET PERFORMANCE/NOBREAK          <- Specifying Not Break
>
```

■ Setting Events

Set events using the SET EVENT command.

The starting/ending point of time measurement and points to measure pass count are specified by events.

Events at 8 points (1 to 8) can be set. However, in the performance measurement, the intervals, starting event number and ending event number are fixed in the following combination.

- **Measuring Time**

  Four intervals have the following fixed event number combination.

Interval	Starting Event Number	Ending Event Number
1	1	2
2	3	4
3	5	6
4	7	8

- **Measuring Count**

  The specified events become performance measurement points automatically.

■ Executing Program

Start measuring when executing a program by using the GO or CALL command. If a break occurs during interval time measurement, the data for this specific interval is discarded.

■ Displaying Performance Measurement Data

Display performance measurement data by using the SHOW PERFORMANCE command.

■ Clearing Performance Measurement Data

Clear performance measurement data by using the CLEAR PERFORMANCE command.

[Example]

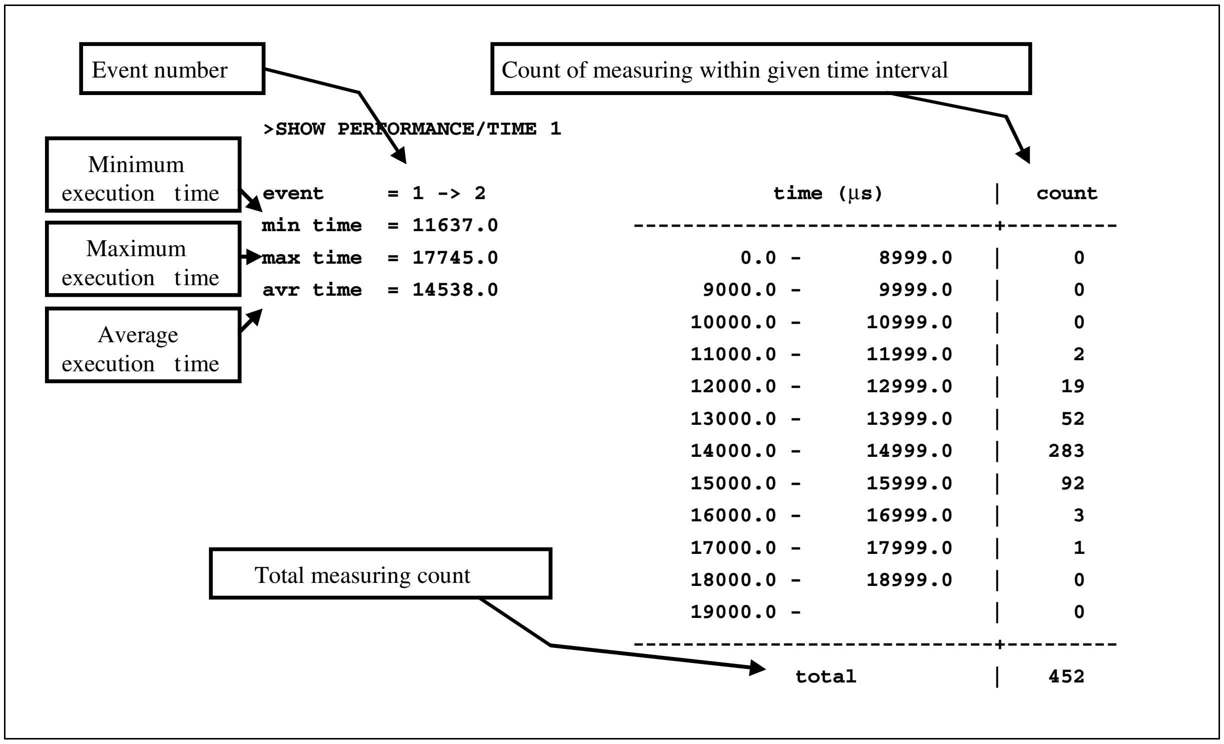
```
>CLEAR PERFORMANCE
>
```

2.2.7.2 Display Performance Measurement Data

Display the measured time and measuring count by using the SHOW PERFORMANCE command.

■ Displaying Measured Time

To display the time measured, specify the starting event number or the ending event number.



```
>SHOW PERFORMANCE/TIME 1

event     = 1 -> 2                    time (µs)         |  count
min time  = 11637.0          ----------------------------+---------
max time  = 17745.0               0.0 -      8999.0     |     0
avr time  = 14538.0            9000.0 -      9999.0     |     0
                              10000.0 -     10999.0     |     0
                              11000.0 -     11999.0     |     2
                              12000.0 -     12999.0     |    19
                              13000.0 -     13999.0     |    52
                              14000.0 -     14999.0     |   283
                              15000.0 -     15999.0     |    92
                              16000.0 -     16999.0     |     3
                              17000.0 -     17999.0     |     1
                              18000.0 -     18999.0     |     0
                              19000.0 -                 |     0
                             ----------------------------+---------
                                    total               |   452
```

The lower time limit, upper time limit and display interval can be specified. The specified time value is in 1µs, when the minimum measurement unit of timer is set to 1 µs by the SET TIMERSCALE command, and in 100 ns when the minimum is set to 100 ns.

```
>SHOW PERFORMANCE/TIME  1,13000,16999,500
event    = 1 -> 2                time (μs)               |  count
min time = 11637.0      ------------------------------+---------
max time = 17745.0             0.0 -      12999.0    |      21
avr time = 14538.0         13000.0 -      13499.0    |      13
                           13500.0 -      13999.0    |      39
                           14000.0 -      14499.0    |     121
                           14500.0 -      14999.0    |     162
                           15000.0 -      15499.0    |      76
                           15500.0 -      15999.0    |      16
                           16000.0 -      16499.0    |       2
                           16500.0 -      16999.0    |       1
                           17000.0 -      17499.0    |       1
                        ------------------------------+---------
                               total                 |     452
```

Lower time limit for display

Upper time limit for display

2.2.8 Measuring Coverage

This emulator has the C0 coverage measurement function. Use this function to find now many percentage of an entire program has been executed.

■ Coverage Measurement Function

When testing a program, the program is executed with various test data input and the results are checked for correctness. When the test is finished, every part of the entire program should have been executed. If any part has not been executed, there is a possibility that the test is insufficient.

This emulator coverage function is used to find now many percentage of the whole program has been executed. In addition, details such as which addresses were not accessed can be checked.

This enables the measurement coverage range to be set and the access attributes to be measured.

To execute the C0 coverage, set a range within the code area and set the attribute to Code attribute. In addition, specifying the Read/Write attribute and setting a range in the data area, permits checking the access status of variables such as finding unused variables, etc.

■ Coverage Measurement Procedures

The procedure for coverage measurement is as follows:

- Set range for coverage measurement:	SET COVERAGE
- Measuring coverage:	GO, STEP, CALL
- Displaying measurement result:	SHOW COVERAGE

■ Coverage Measurement Operation

The following operation can be made in coverage measurement:

- Load/Save of coverage data:	LOAD/COVERAGE, SAVE/COVERAGE
- Abortion and resume of coverage measurement:	ENABLE COVERAGE, DISABLE COVERAGE
- Clearing coverage data:	CLEAR COVERAGE
- Canceling coverage measurement range:	CANCEL COVERAGE

2.2.8.1 Coverage Measurement Procedures

The procedure for coverage measurement is as follows:
- **Set range for coverage measurement: SET COVERAGE**
- **Measure coverage: GO, STEP, CALL**
- **Display measurement result: SHOW COVERAGE**

■ Setting Range for Coverage Measurement

Use the SET COVERAGE command to set the measurement range. Up to 32 ranges can be specified.

By specifying /AUTOMATIC for the command qualifier, the code area for the loaded module is set automatically. However, the library code area is not set when the C compiler library is linked.

[Example]

```
>SET COVERAGE 0FF00..0FFFF
```

■ Measuring Coverage

When preparing for coverage measurement, execute the program.

Measurement starts when the program is executed by using the GO, STEP, or CALL command.

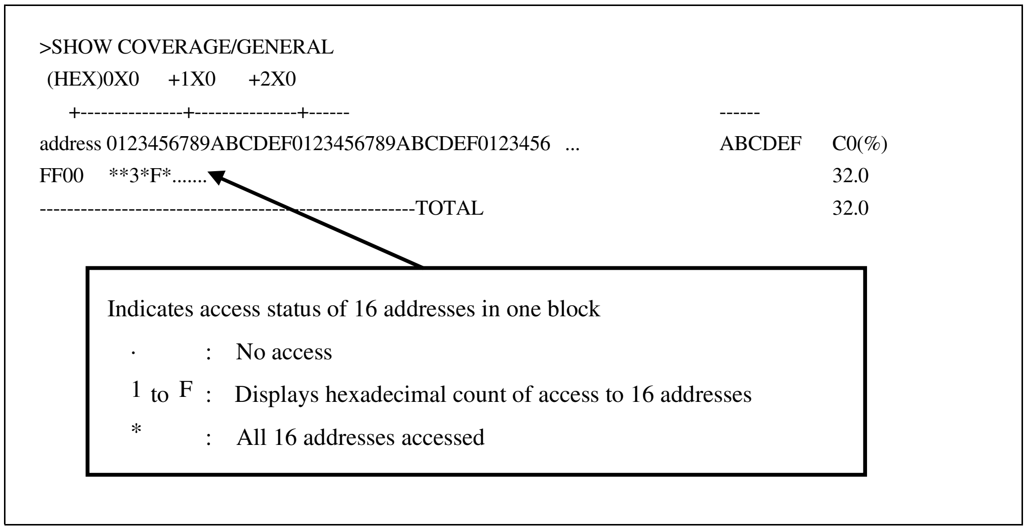
■ Displaying Coverage Measurement Result

To display the coverage measurement result, use the SHOW COVERAGE command. The following can be displayed:

- Coverage ratio of total measurement area
- Summary of 16 addresses as one block
- Details indicating access status of each address
- Coverage Ratio of Total Measurement Area (Specify /TOTAL for command qualifier.)

```
>SHOW COVERAGE/TOTAL
total coverage : 82.3%
```

- Summary (Specify /GENERAL for command qualifier.)

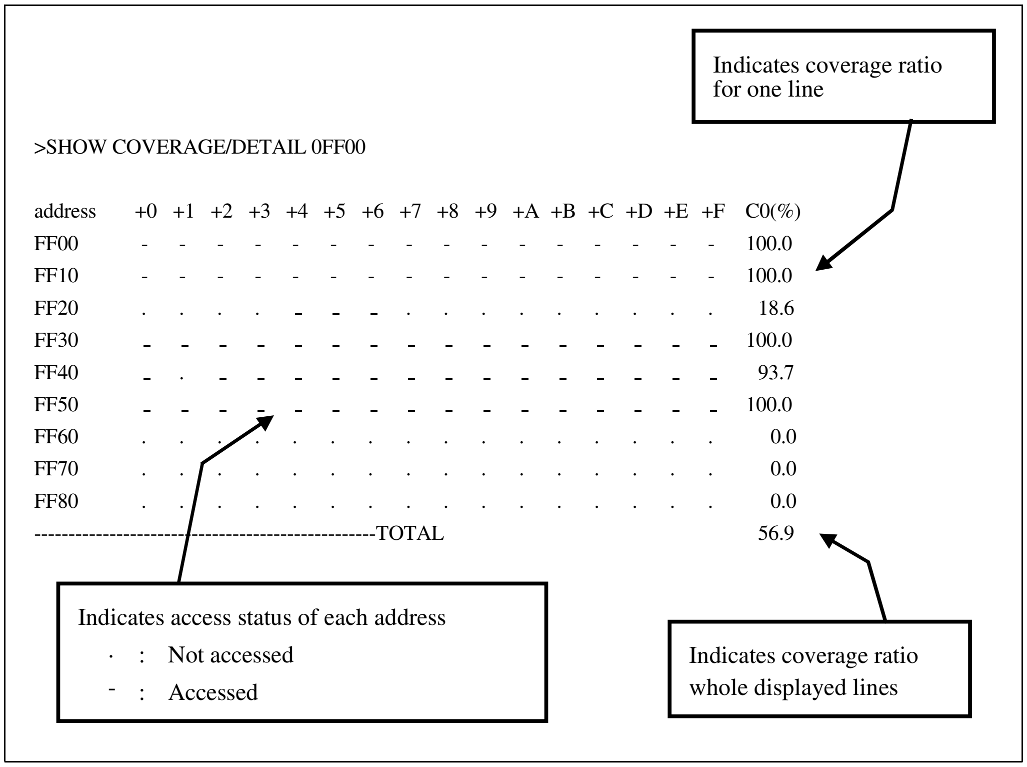


```
>SHOW COVERAGE/GENERAL
 (HEX)0X0     +1X0      +2X0
    +---------------+---------------+------                                ------
address 0123456789ABCDEF0123456789ABCDEF0123456  ...                       ABCDEF     C0(%)
FF00    **3*F*.......                                                                 32.0
-------------------------------------------------------TOTAL                          32.0
```

Indicates access status of 16 addresses in one block

·	:	No access
1 to F	:	Displays hexadecimal count of access to 16 addresses
*	:	All 16 addresses accessed

- Details (Specify /DETAIL for command qualifier.)



Indicates coverage ratio for one line

```
>SHOW COVERAGE/DETAIL 0FF00

address   +0  +1  +2  +3  +4  +5  +6  +7  +8  +9  +A  +B  +C  +D  +E  +F   C0(%)
FF00       -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   100.0
FF10       -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   100.0
FF20       .   .   .   .   -   -   -   .   .   .   .   .   .   .   .   .    18.6
FF30       -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   100.0
FF40       -   .   -   -   -   -   -   -   -   -   -   -   -   -   -   -    93.7
FF50       -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   100.0
FF60       .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .     0.0
FF70       .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .     0.0
FF80       .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .     0.0
------------------------------------------------TOTAL                       56.9
```

Indicates access status of each address

·	:	Not accessed
-	:	Accessed

Indicates coverage ratio whole displayed lines

2.2.9 Measuring Execution Time Using Emulation Timer

The timer for measuring time is called the emulation timer. This timer can measure the time from the start of MCU operation until suspension.

■ Measuring Executing Time Using Emulation Timer

Choose either 1 µs or 100 ns as the minimum measurement unit for the emulation timer and set the measurement unit using the SET TIMERSCALE command.

When 1 µs is selected as the minimum unit, the maximum is about 70 minutes; when 100 ns is selected as the minimum unit, the maximum is about 7 minutes.

The default is 1 µs.

By using this timer, the time from the start of MCU operation until the suspension can be measured.

The measurement result is displayed as two time values: the execution time of the preceding program, and the total execution time of programs executed so far plus the execution time of the preceding program.

If the timer overflows during measurement, a warning message is displayed. Measurement is performed every time a program is executed.

The emulation timer cannot be disabled but the timer value can be cleared instead.

Use the following commands to control the emulation timer.

SHOW TIMER: Displays measured time

CLEAR TIMER: Clear measured timer

[Example]

```
 >GO  main,$25
Break at FF0D by breakpoint
 >SHOW TIMER
<timer>
        from initialize      =    0h 00m 42s 108ms 264ns [Time]
        from last executed   =    0h 00m 03s 623ms 874ns [Time]
        >CLEAR TIMER
         >SHOW TIMER
<timer>
        from initialize      =     0h 00m 00s 000ms 000ns [Time]
        from last executed   =     0h 00m 00s 000ms 000ns [Time]
         >
```

2.2.10 Sampling by External Probe

An external probe can be used to sample (input) data. There are two sampling types: sampling the trace buffer as trace data, and sampling using the SHOW SAMPLING command.

■ Sampling by External Probe

There are two sampling types to sample data using an external probe: sampling the trace buffer as trace data, and sampling using the SHOW SAMPLING command.

When data is sampled as trace data, such data can be displayed by using the SHOW TRACE command or SHOW MULTITRACE command, just as with other trace data. Sampling using the SHOW SAMPLING command, samples data and displays its state.

In addition, by specifying external probe data as events, such events can be used for aborting a program, and as multitrace and performance trigger points.

Events can be set by using the SET EVENT command.

■ External Probe Sampling Timing

Choose one of the following for the sampling timing while executing a program.

- At rising edge of internal clock (clock supplied by emulator)
- At rising edge of external clock (clock input from target)
- At falling edge of external clock (clock input from target)

Use the SET SAMPLING command to set up; to display the setup status use the SHOW SAMPLING command.

When sampling data using the SHOW SAMPLING command, sampling is performed when the command is executed and has nothing to do with the above settings.

[Example]

```
>>SET SAMPLING/INTERNAL
>>SHOW SAMPLING
sampling timing : internal
channel      7 6 5 4 3 2 1 0
             1 1 1 1 0 1 1 1
```

■ Displaying and Setting External Probe Data

When a command that can use external probe data is executed, external probe data is displayed in 8-digit binary or 2-digit hexadecimal format. The displayed bit order is in the order of the IC clip cable color code order (Table 2.2-11). The MSB is at bit 7 (Violet), and the LSB is at bit 0 (Black). The bit represented by 1 means HIGH, while the bit represented by 0 means LOW. When data is input as command parameters, these values are also used for input.

Table 2.2-11 Bit Order of External Probe Data

IC Clip Cable Color	Violet	Blue	Green	Yellow	Orange	Red	Brown	Black
Bit Order	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	External probe data							

■ Commands for External Probe Data

Table 2.2-12 shows the commands that can be used to set or display external probe data.

Table 2.2-12 Commands that can be used External Probe Data

Usable Command	Function
Set Sampling	Sets sampling timing for external probe
Show Sampling	Samples external probe data
Set Event	Enables to specify external probe data as condition for event 1
Show Event	Displays event setup status
Show Trace	Displays external probe trace-sampled (single trace)
Show Multitrace	Displays external probe trace-sampled (multi-trace)

2.3 Emulator Debugger (MB2146-09)

This section describes the functions of the emulator debugger for the F²MC-8FX family.

■ Emulator Debugger

When choosing the emulator debugger from the setup wizard, select MB2146-09 using ICEs (in-circuit emulators) type.

MB2141

MB2146-09

The emulator debugger (emulator) for the MB2146-09 ICE is software that controls an in-circuit emulator (ICE) from a host computer via a communications line (USB) to evaluate programs.

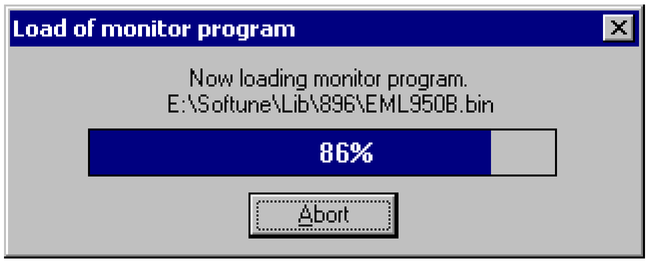
The following series can be debugged:

When using MB2146-09 (USB adapter)

F^2MC-8FX

When the MB2146-09 is used, refer to Appendix D, Setting USB Interface in the SOFTUNE WORKBENCH Operation Manual to set up the USB interface.

When the debugger starts immediately after the power of the user system is turned on, the monitor program is automatically loaded. The following dialog is displayed at loading the monitor program. Once it is loaded, it cannot be reloaded unless the power is turned off. Thus, the monitor program is placed in Lib\896 under the directory installed the Workbench.



Now loading monitor program.
E:\Softune\Lib\896\EML950B.bin

2.3.1 Setting Operating Environment

To operate the emulator, set the operating environment by specifying the clock-up mode and main clock oscillation.
Note that predefined default settings for all these setup items are enabled at startup. Therefore, setup is not required when using the default settings.

■ Setting Operating Environment

For the emulator of the MB2146-09 ICEs, it is necessary to set the clock-up mode and frequency of main oscillation clock (F_{CH}). Note that predefined default settings for all these setup items are enabled at startup. Therefore, setup is not required when using the default settings. Furthermore, the specified setting value can be used for the subsequent default.

● Clock-up mode

The MB2146-09 and communication speed of user system change by the operating frequency of the target MCU. When the operating frequency is reduced, especially in the sub clock mode, the communication speed is also reduced. In this case, optimize the communication speed, the function increasing the operating frequency automatically is called clock-up mode. The default is ON at that time.

Also, this setting is performed at the setup wizard or [Environment]-[Setting of debugging Environment]-[Response Speed] Tab.

Note:

- When the clock-up mode is used, the operating frequency is changed automatically at breaking. If the failure is caused by changing the operating frequency, disables the clock-up modes.
- If a break occurs immediately after changing the system clock mode by the user program, no clock up is performed during oscillations stabilization wait state. Clock up will be performed when oscillations are stabilized.

● Main Clock Oscillation

The MB2146-09 and communication speed of user system change by the operation frequency of the target MCU. The setting of the main clock oscillation (F_{CH}) is required to calculate the operating speed of the target MCU. The default is the maximum frequency that specified MCU operates in the main clock.

Also, this setting is performed at the setup wizard.

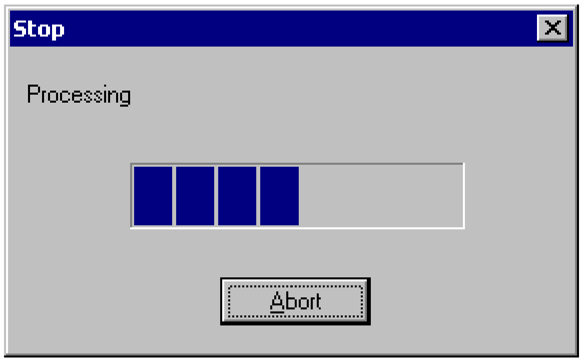
2.3.2 Programming to Flash ROM area

This emulator supports programming to the Flash ROM area.

■ Erasing/Programming Flash ROM

Writing to Flash ROM/code break (software break) functions are supported. The content of Flash ROM area is secured in the buffer within the debugger, and the content of the buffer is referred at reading/writing. Writing to Flash ROM is automatically performed prior to executive operation or reset processing. The programming to Flash ROM can be also performed manually.

The following dialog is displayed at writing to Flash ROM.



The command that programming to Flash ROM is generated has the following;

- SET MEMORY
- SET BREAK
- LOAD
- FILL
- MOVE
- ASSEMBLE

There are the following three functions for the operation of Flash ROM:

1. Updating FLASH memory

   ([Environment] - [FLASH area control] - [Download FLASH memory] menu).

   Updates Flash memory. Flash memory is usually updated automatically prior to executive operation or reset processing. Use this menu when updating Flash memory before this automatic updating.

   This menu is enabled when data in the Flash memory area is changed, requiring the writing to of Flash memory.

2. Uploading FLASH memory

   ([Environment] - [FLASH area control] - [Upload FLASH memory] menu).

   Synchronizes Flash memory and the buffers within the Debugger. Be sure to perform this synchronization when Flash memory is rewritten (updated) by the user program, or the program would not operate properly.

3. Erasing FLASH memory

   ([Environment] - [FLASH area control] - [Erase FLASH memory] menu).

   Erase all data in Flash memory. Note that this operation will erase all code break (software break) settings.

2.3.3 Break

The emulator can use the break function below;
- **Code break**
- **Data break**
- **Monitoring data break**
- **Sequence break**

■ Setup of Code Break (Software Break)

The code break (software break) can be set 256 points.

The command is set as follows;

```
SET BREAK
```

■ Setup of Data Break

The data break is a function to suspend the program execution, when the data access (read or write) is performed to the specified address during the program execution.

The emulator can be set up to 2.

When set as monitoring data break, the data access is not broken.

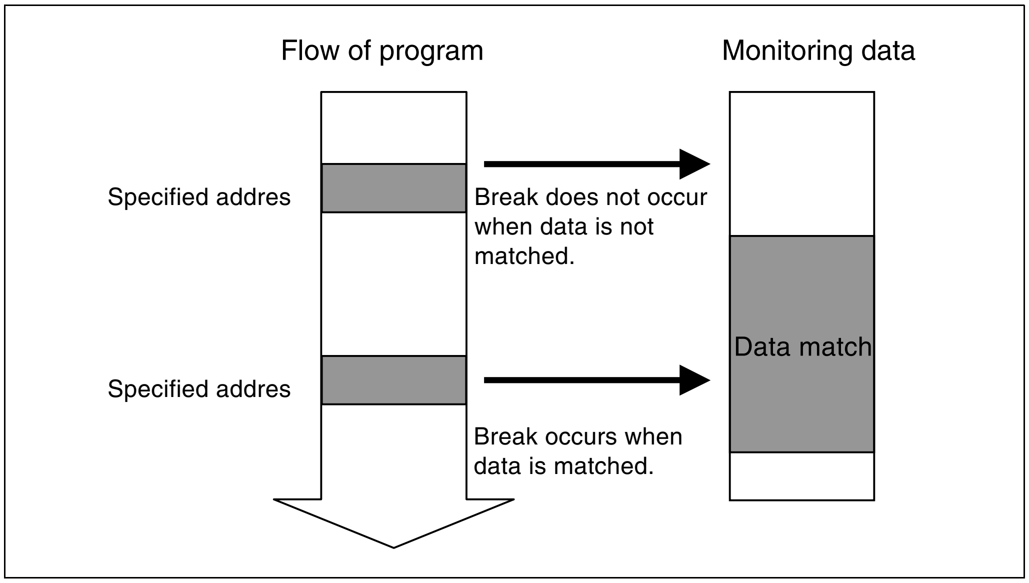
The command is set as follows;

```
SET DATABREAK
```

■ Setup of Monitoring Data Break

The monitoring data break is a special break function when the program is reached the specified address during matching with specified data.

The following figure is displayed the break condition.


Flow of program
Monitoring data
Specified addres
Break does not occur when data is not matched.
Data match
Specified addres
Break occurs when data is matched.

Monitoring data break can be set 1 point.

The command is set the following procedure.

1. SET DATABREAK
2. SET BREAK/DATAWATCH

■ Setup of Sequence Break

The sequence break is a function to suspend the program execution when the program is executed at level 1 and then at level 2 for the specified two addresses.

The emulator can be set only 1.

The command is set below.

```
SET BREAK /SEQUENCE
```

2.3.4 Real-time Trace

While execution a program, the executed address information is sampled and stored in the trace buffer. This function is called "trace".

■ Trace

The program execution history can be analyzed from the data stored by the trace.

Since the trace buffer has a ring structure, when it becomes full, it automatically returns to the start to overwrite existing data.

■ Trace Data

The stored data sampled by the trace is called trace data.

For the emulator debugger of the MB2146-09, 16 divergences immediately before the execution interruption can be sampled.

Note:

When 4096 or more branch instructions are not executed, only 4096 instructions from the branch destination address is displayed.

■ Sampling Trace Data

When the trace function is enabled, the data is sampled during command execution, and it is stored in the trace buffer.

When the program execution is stopped by the break cause such as break point, the trace sampling is ended.

■ Frame number

A number is assigned to each frame of sampled trace data. This number is called a frame number.

The frame number is used to specify the display start position of the trace buffer. The value 0 is assigned to trace data at the position for current program counter (PC). Negative values are assigned to previous trace data.

2.3.4.1 Displaying Trace Data

The data stored in the trace buffer is displayed.

■ Displaying Trace Data Storage Information

Trace window is displayed how much trace data is stored in the trace buffer. Also, the command displays by SHOW TRACE/STATUS.

■ Display Format of Trace Data

There are two types of display format for the trace buffer.

- Display instruction execution only (Display instruction)
  Display the instruction execution in disassembly unit.
- Display in source line units (Display source)
  Display the source line only.

■ Clearing Trace Data

When the trace data is cleared, execute [Clear] within the shortcut menu in the trace window. Also, the command executes the CLEAR TRACE command.

2.3.4.2 Precaution of Trace

This section notes related to display the trace data.

■ Precaution of Trace Function

When the emulator debugger for the MB2146-09 is used, the address information is outputted at the branch instruction fetch, the trace is implemented.

At that time, notes the following points related to display trace data.

The disassembly display is performed after reading from the memory. In this case, when the instruction is completed to write after the code fetch, it does not display correctly.

2.3.5 Notes on Executing Program

This emulater notes the following points.

■ Break at Standby Mode

When the suspend operation is executed in the standby mode, the debugger cancels the standby mode and suspends the execution. Therefore, it is suspended in next address of instruction to be transmitted to the standby mode.

2.4 Abortion of Program Execution (SIM, EML)

The abortion of program execution uses both the simulator and emulator for the F^2MC-8L family.
When program execution is aborted, the address where the break occurred and the break source are displayed.

■ Abortion of Program Execution

When program execution is aborted, the address where the break occurred and the break source are displayed.

In the emulator, debugger the following sources can abort program execution.

- Instruction Execution Breaks
- Monitoring data Breaks
- Data Access Breaks
- Sequential Break
- Guarded Access Breaks
- Trace-Buffer-Full Break
- Performance-Buffer-Full Break
- Forced Break

In the simulator, debugger the following sources can abort program execution.

- Instruction Execution Breaks
- Data Access Breaks
- Guarded Access Breaks
- Trace-Buffer-Full Break
- Forced Break

2.4.1 Instruction Execution Breaks (SIM, EML)

An instruction execution break is a function to let an instruction break through monitoring bus, the chip built-in break points, etc.

■ Instruction Execution Breaks

Use the following commands to control instruction execution breaks.

- SET BREAK: Sets break point
- SHOW BREAK: Displays current break point setup status
- CANCEL BREAK: Cancels break point
- ENABLE BREAK: Enables break point
- DISABLE BREAK: Temporarily cancels break point

When a break occurs due to an instruction execution break, the following message is displayed.

```
Break at  Address  by breakpoint
```

The maximum count of break points that can be set is 65535 both in the simulator and emulator (MB2141), and 256 in the emulator (MB2146-09).

■ Notes on Instruction Execution Breaks (MB2141)

A break occurs before executing the instruction if the break point is set immediately after the instructions listed in Figure 2.4-1 . The debugger is designed to perform step execution internally, and then to break after such execution. Therefore, the last instruction is not executed in real-time.

Figure 2.4-1 Instructions affecting instruction execution breaks

ADDC	A,@EP	ADDC	A, Ri	ADDC	A	ADDCW	A
AND	A,@EP	AND	A, Ri	AND	A	ANDW	A
CALLV	#n	CMP	A,@EP	CMP	A, Ri	CMP	A
CMPW	A	DAA		DAS		DEC	Ri
DECW	A	DECW	EP	DECW	IX	DECW	SP
DIVU	A	INC	Ri	INCW	A	INCW	EP
INCW	IX	INCW	SP	MOV	@A,T	MOV	@EP,A
MOV	A, A@	MOV	A,@EP	MOV	A, Ri	MOV	Ri, A
MOVW	@A, T	MOVW	A, @A	MOVW	A,@EP	MOVW	A, EP
MOVW	A, IX	MOVW	A,PC	MOVW	A, PS	MOVW	SP,A
MOVW	EP,A	MOVW	IX,A	MOVW	PS,A	MOVW	SP,A
MULU	A	OR	A, @EP	OR	A,Ri	OR	A
ORW	A	POPW	A	POPW	IX	PUSHW	A
PUSHW	IX	ROLC	A	RORC	A	SUBC	A,@EP
SUBC	A,Ri	SUBC	A	SUBCW	A	SWAP	
XCH	A, T	XCHW	A,EP	XCHW	A,IX	XCHW	A,SP
XCHW	A, T	XOR	A,@EP	XOR	A,Ri	XOR	A
XORW	A						

If an instruction execution break is set following the 1-byte branch instruction shown below, it occurs immediately after the instruction is executed, because the 1-byte branch instruction is affected by prefetch of the next instruction when executed. Instructions when the instruction execution break is set are just prefetched but not executed.

RET	RET1	JMP@A	CALLV#vct

To avoid this, set the instruction execution break shifted one byte or set a breakpoint using the SET EVENT/CODE command, which is unaffected by prefetch.

2.4.2 Monitoring Data Break (MB2146-09)

A monitoring data break is a function, when program execution at a specified address, is completed to abort a running program when data access (Read or Write) is made to the specified data area address while the program is executing.

■ Monitoring Data Breaks

A data monitor break is a function, when program execution at a specified address (However, before execution of target instruction), to abort an executing program when the MCU access data at the specified address.

Monitoring data breaks can be controlled using the following commands;

SET BREAK/DATAWATCH:	Sets break point
SHOW BREAK/DATAWATCH:	Display current break point setup status
CANCEL BREAK/DATAWATCH:	Cancel break point
ENABLE BREAK/DATAWATCH:	Enables break point
DISABLE BREAK/DATAWATCH:	Temporarily cancels break point

When a break occurs due to a monitoring data break, the following message is displayed;

```
Break at address by datawatch
```

The maximum count of break points are as follows;

MB2146-09 Max: 1 point

Note:

This is specific function in the MB2146-09 ICE emulator.

2.4.3 Data Access Breaks (SIM, EML)

A data access break is a function to abort a executing program when data access (Read or Write) is made to the specified address while the program is executing.

■ Data Access Breaks

A data access break is a function to suspend a running program when the MCU accesses data at the specified address.

Data access breaks can be controlled using the following commands:

- SET DATABREAK: Sets break points
- SHOW DATABREAK: Displays current break point setup status
- CANCEL DATABREAK: Cancels break point
- ENABLE DATABREAK: Enables break points
- DISABLE DATABREAK: Temporarily cancels break points

When a break occurs due to a data access break, the following message is displayed:

```
Break at  Address  by databreak at  Access address
```

The maximum count of break points that can be set is 65535 both in the simulator and emulator (MB2141), and 2 in the emulator (MB2146-09).

2.4.4 Sequential Break (EML)

A sequential break is a function to suspend a executing program, when the sequential condition is met by event sequential control.

■ Sequential Break

Use a sequential break when the event mode is set to normal mode using the SET MODE command. Set a sequential break as follows:

Set as the following procedure in emulator (MB2141).

1. Set event mode (SET MODE).
2. Set events (SET EVENT).
3. Set sequencer (SET SEQUENCE).

Set as the following procedure in emulator (MB2146-09).

SET BREAK/SEQUENCE

When a break occurs due to a sequential break, the following message is displayed:

```
Break at  Address  by sequential break (level = Level No.)
```

2.4.5 Guarded Access Breaks (SIM, EML)

A guarded access break aborts a executing program when access is made in violation of the access attribute set by using the [Setup]-[Memory Map] command, and access is attempted to a guarded area (access-disabled area in undefined area).

■ Guarded Access Breaks

Guarded access breaks are as follows:

Code Guarded

An instruction has been executed for an area having no code attribute.

Read Guarded

A read has been attempted from the area having no read attribute.

Write Guarded

A write has been attempted to an area having no write attribute.

If a guarded access occurs while executing a program, the following message is displayed on the Status Bar and the program is aborted.

```
Break at  Address  by guarded access {code/read/write} at  Access address
```

Not supported in the emulator (MB2146-09).

2.4.6 Trace-Buffer-Full Break (SIM, EML)

A trace-buffer-full break occurs when the trace buffer becomes full.

■ Trace-Buffer-Full Break

To set a trace-buffer-full break, use the [Setup]-[Trace] command in the short-cut menu of the [Analyze]-[Trace] command, or use the SET TRACE/BREAK command.

When a break occurs due to a trace-buffer-full break, the following message is displayed:

```
Break at  Address  by  trace buffer full
```

Note:

Not supported in the emulator (MB2146-09).

2.4.7 Performance-Buffer-Full Break (EML)

A performance-buffer-full break is a function to abort a executing program when the buffer for storing performance measurement data becomes full.

■ Performance-Buffer-Full Break

To set a performance-buffer-full break, use the SET PERFORMANCE command. If a performance-buffer-full break is not specified, no break occurs even when the performance buffer becomes full.

When a break occurs due to a performance-buffer-full break, the following message is displayed:

```
Break at  Address  by performance buffer full
```

Note:

Not supported in the emulator (MB2146-09).

2.4.8 Forced Break (SIM, EML)

A executing program can be forcibly aborted by using the [Run]-[Abort] command.

■ Forced Break

When a break occurs due to a forced break, the following message is displayed.

```
Break at  Address  by command abort request
```

■ Forced Break in Power-Save Consumption Mode and Hold State (MB2141)

A forced break is not allowed in the emulator while the MCU is in the power-save consumption mode or hold state. When a forced break is requested by the [Run]-[Abort] command while executing a program, the command is disregarded if the MCU is in the power-save consumption mode or hold state. If a break must occur, then reset the cause at user system side, or reset the cause by using the [Run]-[Reset of MCU] command, after inputting the [Run]-[Abort] command.

When the MCU enters the power-save consumption mode or hold state while executing, the status is displayed on the Status Bar.

CM25-00324-1E

FUJITSU SEMICONDUCTOR • CONTROLLER MANUAL

F^{2}MC^{TM}-8L/8FX FAMILY

SOFTUNE^{TM} WORKBENCH

USER'S MANUAL

April 2004 the first edition

Published **FUJITSU LIMITED** Electronic Devices

Edited Business Promotion Dept.